cadence™

Virtuoso® Analog Design Environment L SKILL Reference

Product Version 6.1.6
October 2014

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

Contents

9
Analysis Functions

Preface

Important

Unless stated otherwise, the functions included in this manual are applicable to both ICADV12.1 and IC6.1.6.

This manual describes the SKILL functions that you can use with Virtuoso Analog Design Environment L. You can use these functions to modify Virtuoso Analog Design Environment L to better suit your needs. You can also use these functions to help you integrate tools or simulators into Virtuoso Analog Design Environment L.

This manual assumes you are familiar with the Cadence SKILL™ language.

The preface discusses the following:

Scope of this Manual

The SKILL functions described in this manual can be used in either IC6.1.6, ICADV12.1, or both of these releases. Functions that are supported only in a particular release are identified using the **(ICADV12.1 ONLY)** or **(IC6.1.6 ONLY)** text at the beginning of the function description. All other functions are supported in both releases.

Important

Only the functions and arguments described in this manual are supported for public use. All other functions, and undocumented aspects of the functions described here, are private and subject to change at any time.

Related Documents for Virtuoso Analog Design Environment L SKILL Functions

The SKILL programming language is often used with other Virtuoso products or requires knowledge of a special language. The following documents give you more information about these tools and languages.

- If you want to use the SKILL language functions, the Virtuoso SKILL++™ functions, and the SKILL++ object system (for object-oriented programming), you need to read the *Cadence SKILL Language User Guide*.
- If you want to see descriptions, syntax, and examples for the SKILL and SKILL++ functions, you need to read the *Cadence SKILL Language Reference*.
- If you want to see descriptions, syntax, and examples for the object system functions, you need to read the *Cadence SKILL++ Object System Reference*.
- *Virtuoso Design Environment SKILL Functions Reference* provides detailed information about the SKILL functions that interface to applications in the Virtuoso Design Environment.
- If you want to design and simulate analog and mixed-signal circuits, you need to read the *VirtuosoAnalog Design Environment L User Guide*.
- If you want to set up, simulate, and analyze circuit data without starting the Virtuoso analog design environment, you need to read the *OCEAN Reference*.
- If you want to use the Virtuoso®schematic editor to enter your design, you need to read the *Virtuoso Schematic Editor L User Guide*.

- If you want to use the Virtuoso®layout editor to enter your design, you need to read the *Virtuoso Layout Suite L User Guide*.

Typographic and Syntax Conventions

This list describes the syntax conventions used for the Virtuoso analog design environment SKILL functions.

`literal`	Nonitalic words indicate keywords that you must enter literally. These keywords represent command (function, routine) or option names.
`argument` (*`z_argument`*)	Words in italics indicate user-defined arguments for which you must substitute a name or a value. (The characters before the underscore (_) in the word indicate the data types that this argument can take. Names are case sensitive. Do not type the underscore (*`z_`*) before your arguments.)
|	Vertical bars (OR-bars) separate possible choices for a single argument. They take precedence over any other character.
`[ ]`	Brackets denote optional arguments. When used with OR-bars, they enclose a list of choices. You can choose one argument from the list.
`{ }`	Braces are used with OR-bars and enclose a list of choices. You must choose one argument from the list.
`...`	Three dots (`...`) indicate that you can repeat the previous argument. If you use them with brackets, you can specify zero or more arguments. If they are used without brackets, you must specify at least one argument, but you can specify more.
`argument...`	specify at least one, but more are possible
`[argument]...`	you can specify zero or more
`,...`	A comma and three dots together indicate that if you specify more than one argument, you must separate those arguments by commas.

`=>`	A right arrow points to the possible values that a SKILL function can return. This character is represented by an equal sign and a greater than sign.
`/`	A slash separates the possible values that can be returned by a SKILL function.
`<`*`yourSimulator`*`>`	Angle brackets are used to indicate places where you need to insert the name of your simulator. Do not include the angle brackets when you insert the simulator name.

The language requires any characters not included in the list above. You must enter required characters literally.

SKILL Syntax Examples

The following examples show typical syntax characters used in the SKILL language.

Example 1

```
list( g_arg1 [g_arg2] ...
     )
     => l_result
```

Example 1 illustrates the following syntax characters.

`list`	Plain type indicates words that you must enter literally.
`g_arg1`	Words in italics indicate arguments for which you must substitute a name or a value.
`( )`	Parentheses separate names of functions from their arguments.
`_`	An underscore separates an argument type (left) from an argument name (right).
`[ ]`	Brackets indicate that the enclosed argument is optional.
`=>`	A right arrow points to the return values of the function. Also used in code examples in SKILL manuals.

`...`	Three dots indicate that the preceding item can appear any number of times.

Example 2

```
needNCells(
     s_cellType | st_userType
     x_cellCount
     )
     => t/nil
```

Example 2 illustrates two additional syntax characters.

`|`	Vertical bars separate a choice of required options.
`/`	Slashes separate possible return values.

Identifiers Used to Denote Data Types

The Cadence SKILL language supports different data types to identify the type of value you can assign to an argument.

Data types are identified by a single letter followed by an underscore; for example, *t* is the data type in *t_viewNames* and denotes that the argument in question accepts a character string. Data types and the underscore are used as identifiers only; they should not be typed.

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

Prefix	Internal Name	Data Type
G	gdmSpecIlUserType	gdm spec
h	hdbobject	hierarchical database configuration object
K	mapiobject	MAPI object
l	list	linked list
L	tc	Technology file time stamp
m	nmpIlUserType	nmpIl user type
M	cdsEvalObject	—
n	number	integer or floating-point number
o	userType	user-defined type (other)
p	port	I/O port
q	gdmspecListIlUserType	gdm spec list
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	—
w	wtype	window type
x	integer	integer number
y	binary	binary function
&	pointer	pointer type

Additional Learning Resources

Cadence provides various Rapid Adoption Kits that you can use to learn how to employ Virtuoso applications in your design flows. These kits contain workshop databases, designs, and instructions to run the design flow.

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

- SKILL Language Programming Introduction
- SKILL Language Programming
- Advanced SKILL Language Programming

For further information on the training courses available in your region, visit the Cadence Training portal. You can also write to training_enroll@cadence.com.

Note: The links in this section open in a new browser. They initially display the requested training information for North America, but if required, you can navigate to the courses available in other regions.

1

Introduction

The Virtuoso Analog Design Environment L SKILL functions in this manual are organized into chapters according to their purpose. Within each chapter, the functions are in alphabetical order.

You can use the functions in this manual to do the following:

- Change the values of existing variables, such as the default values of model paths or simulator options.

  For more information, see "Tools and Sessions" on page 35.

- Help you integrate a simulator into the Virtuoso analog design environment. An integrator is someone responsible for making the changes necessary to include a circuit in the Virtuoso analog design environment (You need to purchase the OASIS product in order to integrate a simulator.)

- Add new features to existing simulators.

  For more information, see "Adding Features to Simulators" on page 42.

If you need to change the menus for the Simulation window, see "Changing Virtuoso Analog Design Environment Banner Menus" on page 44.

Note: Read this entire chapter before you use any of the functions in this manual.

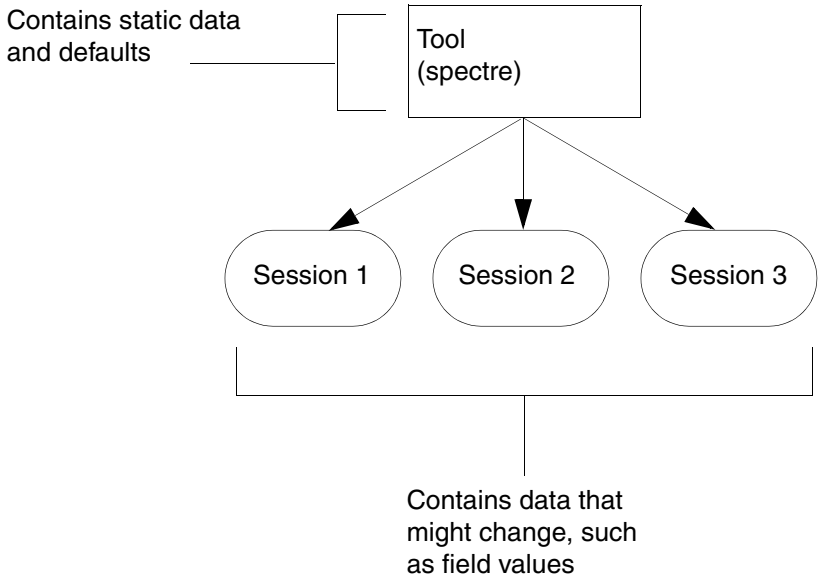
Cadence has integrated simulators into the analog design environment by using the direct simulation approach. With direct simulation, the netlist uses the syntax of the simulator you are using, without any processing to evaluate expressions. The passed parameters, design variables, functions, and so on are all resolved by the simulator. The netlist is a direct reflection of the design.

Tools and Sessions

Understanding the difference between a tool object and a session object will help you use many of the routines in this manual. A *tool* object is a data structure that contains all the

default and static data for a simulator. For example, the fields on an analysis form are static (a transient analysis always has a *From, To,* and *By* field). A *session* object contains data that might change between simulator uses, such as the values of the *From*, *To*, and *By* fields, or design variables or model paths.

Several sessions can be associated with one tool. This means you can start several Cadence sessions, each with different transient analysis settings, for example. However, all these sessions are associated with one tool (`spectree`).



Most of the routines in this manual take the `tool` argument. Some take either the `tool` or the `session` argument, and some take only the `session` argument. For example, `asiSetEnvOptionVal` takes the following arguments:

```
asiSetEnvOptionVal( {o_tool | o_session} s_name g_value)
```

This means you can pass in either a tool or a session object. If you pass in a tool object, you are modifying the tool and every session created *after* this. If you pass in a session object, your changes affect the current session only.

Example

Suppose you have a `cdsSpice` tool with the model path set to `~/models`, and a schematic window open with your design. When you open a Simulation window, which is set to use the `cdsSpice` simulator by default, you are starting a `cdsSpice` session. If you bring up the Environment Options form, the model path is set to `~/models`. You can change the model path for the tool by typing the following in the CIW:

```
asiSetEnvOptionVal( asiGetTool('cdsSpice) 'modelPath
"~/processA/best/models")
```

Now if you check the model path from your existing `cdsSpice` simulation window, it is still set to `~/models`. This is because this session was previously created based on the original tool. However, if you open a second `cdsSpice` simulation window, a new session is created based on the modified `cdsSpice` tool. In this session, the model path is set to `~/processA/best/models`.

If you do not want to open another session, but you want to change the model path in the current session, you might type the following command in the CIW:

```
asiSetEnvOptionVal( asiGetCurrentSession() 'modelPath
"~/processA/best/models")
```

This changes the model path in the existing session only. If you open another session, the `modelPath` value is still set to `~/models`. This is because the command did not change the default value of the model path, it changed the session value only.

You can think of a tool as the template from which sessions are created. Changing a tool affects all sessions created after the tool is changed, but does not affect existing sessions.

Note: The functions in this manual are contained primarily in the `oasis.cxt` and `analog.cxt` files. You must load these context files before using the functions. (Bringing up the Simulation window automatically loads these contexts.)

Use Models

There are two common use models for the routines that take the tool and/or session object.

Working with Tools

Often you need to modify an item or option related to a tool. These types of operations might include the following:

- Adding an option to a tool
- Changing a tool option
- Deleting a tool option
- Setting a tool option's value
- Getting a tool option (object)

You can set up a routine in your `.cdsinit` file to do this. For example, suppose you wanted to set the following options for every new `spectreS` session.

- Model path set to `~/processA/best/models`
- Include file set to `~/processA/best/models/include`

You might add the following routines to your `.cdsinit` file:

```
asiSetEnvOptionVal( asiGetTool('spectreS) 'modelPath
"~/processA/best/models")
asiSetEnvOptionVal( asiGetTool('spectreS) 'includeFile
"~/processA/best/models/include")
```

Because these calls to `asiSetEnvOptionVal` are in the `.cdsinit` (before any sessions are opened), all subsequent sessions contain the specified model path and include file.

Getting Values from Sessions

Sometimes you want to get a value from an existing session.

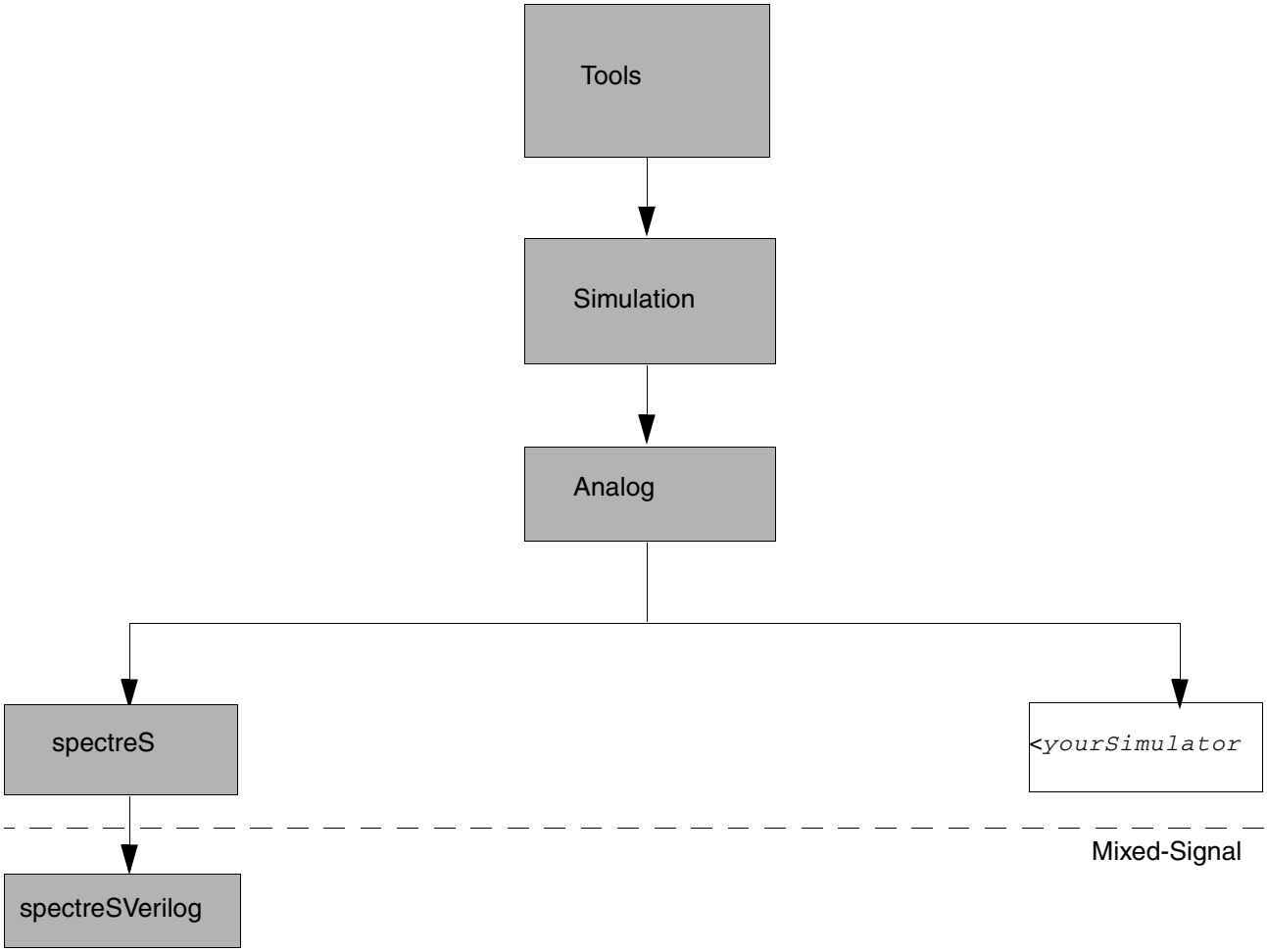
For example, suppose you are writing some custom SKILL code in which you need to get the value of the transient *to* field for the existing session. In this case, you might use the following code:

```
session = asiGetCurrentSession()
analysis = asiGetAnalysis(session 'tran)
valueOfTo=asiGetAnalysisFieldVal( analysis 'to)
```

Class Structures

In Virtuoso analog design environment, much of the underlying code utilizes *classes*. A class determines the structure and behavior of its instances, which are known as objects. A class inherits information from its parent class (or super class) and passes information to its subclass. Understanding classes will help you use some of the functions in this manual that utilize the inheritance mechanism. (These functions are called methods and are explained in the section Methods in the Virtuoso Analog Design Environment on page 40“.)

The following figure shows how the classes are organized in Virtuoso analog design environment.



The class structure is organized as follows:

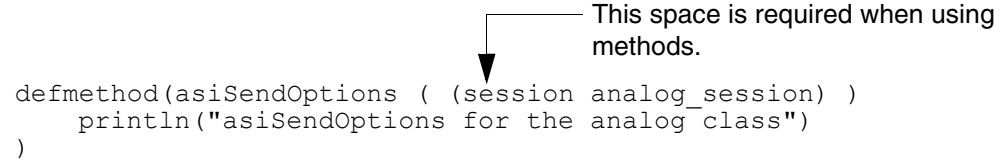
- The *Tools Class* is the base or root class. Anything declared in the Tools Class is inherited by all the other subclasses.
- *The Simulation Class* inherits everything from the Tools Class. Simulation is the base class for all simulators, and anything declared in the Simulation Class is inherited by all the simulators.
- *The Analog Class* inherits everything from the Simulation Class. The Analog Class adds all the generic analog simulation features to the set. Anything declared in the Analog Class is inherited by all the analog simulators.

If an inherited feature from any level is not applicable to a lower level, you can delete or change the feature at the lower level.

Methods in the Virtuoso Analog Design Environment

Some of the functions in this manual are defined as *methods* for integrators. A method is a function that can be *overloaded*. In other words, the same function can be defined in different ways depending on the class of the arguments that users pass to the function. With methods, the same function name can specify a general class of actions. The user, by supplying an initial argument of a particular class to the function, determines the specific action of the function. (The SKILL++ object system analyzes the class of the initial argument to determine which variant of the function to call.) The methods in this manual have examples for integrators.

When using methods, the space after the name of the procedure is required. (When using SKILL procedures, the space after the name of the procedure is not allowed.)



You can use `defmethod` to create a method that is specialized for a particular class. If a user calls a function with an initial argument of this class, the SKILL++ object system looks for the corresponding method for that class. If there isn't a match for that class, the SKILL++ object system looks for a match in the class above, and so on.

Using methods gives you the following advantages:

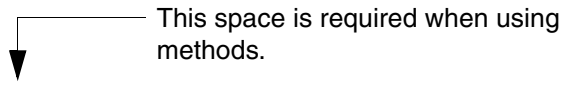
- You can overload an existing method for your own simulator-specific classes. In other words, you can customize an existing Virtuoso analog design environment function for your simulator.
- You can use the `callNextMethod` procedure to use an inherited method and add some code before or after it.

For more information about methods, consult the Cadence SKILL++ Object System Reference.

Overloading Methods

You can overload an existing method for your own simulator-specific classes. For example, there is a method to send simulation options (`asiSendOptions`) in the Analog Class. You probably need to create a different method to send your options. To do this, create your own method named `asiSendOptions` that is used by your simulator. You do not have to modify the routine that calls `asiSendOptions`. Your procedure takes precedence over the method in the Analog Class.

For example, the following is an Analog Class method declaration to send options:




```
defmethod(asiSendOptions ( (session analog_session) )
    println("asiSendOptions for the analog class")
)
```

To create your own `asiSendOptions` procedure, use the following declaration:

```
defmethod(asiSendOptions ( (session <yourSimulator>_session) )
    println("asiSendOptions for your simulator class")
)
```

When Virtuoso analog design environment sends the simulation options for your simulator, your `asiSendOptions` method is called because the first argument is of the class *<yourSimulator>*_session. For more information about *defmethod*, consult the Cadence SKILL++ Object System Reference.

Note: You should replace *<yourSimulator>* with the name of your simulator. Do not include the angle brackets (<>).

Customizing an Inherited Method for Your Simulator

The `callNextMethod` procedure lets you use an inherited method and add some code before or after it.

For example, suppose you need to modify the transient analysis to add transient options for your simulator. You need to write a method to send the information to Cadence SPICE. There is already a method defined to send a transient analysis to Cadence SPICE. Because there are no transient options in the default analysis, the code does not send options. So you can use `callNextMethod()` to call the method to format the analysis, then call your procedure to format the options.

```
defmethod(asiFormatAnalysis (
    (analysis <yourSimulator>_tran_analysis) fp)
    callNextMethod()
    asiFormatAnalysisOption(analysis fp)
```

```
    t
)
```

In this example, `callNextMethod` calls the analog `asiFormatAnalysis` method to format the transient analysis and send it to Cadence SPICE. Then your `asiFormatAnalysis` method calls `asiFormatAnalysisOption` to format and send the options.

For more information about *callNextMethod*, consult the Cadence SKILL++ Object System Reference.

Adding Features to Simulators

You can use the functions in this manual to add new features to existing simulators. Create a SKILL file to hold your information.

1. Use the `asiGetTool` function to get the tool object associated with the simulator you want to modify.
2. Add the features you want by using the procedures in this manual and passing in the *tool* argument.

   For example, you might use `asiAddEnvOption` to add an environment option or `asiAddSimOption` to add a simulator option.
3. If needed, add the code to send your option to Cadence SPICE (to send to your target simulator).
4. If needed, add the code to invalidate the flowchart step if the value of your option changes.
5. Load your code changes into your `.cdsinit` file.

   Later, you can add your changes to your site `.cdsinit` file.
6. Add any necessary changes to the `.cdsenv` file.

   - Type `virtuoso` in a UNIX shell.
   - Type the following command in the CIW and substitute the appropriate simulator name for *<simulator>*. Do not include the angle brackets (<>).

   ```
   asiCreateCdsenvFile('<simulator>)
   ```

   A file called *<simulator>*`CdsenvFile` is created in your current working directory.

- Use this file to replace the existing `.cdsenv` file in the *<install_dir>*`/tools/dfII/etc/tools/<`*simulator*`>` directory. Be sure to save a backup copy of the existing `.cdsenv` file before completing this step. Use the following command to replace the existing `.cdsenv` file with your new file:

  ```
  mv <simulator>CdsenvFile <install_dir>/tools/dfII/etc/
      tools/<simulator>/.cdsenv
  ```

Example

The following example shows how you might add an environment option called `myFile` to the environment options form for spectreS. The steps from the previous procedure are called out in the comments.

```
; get the spectreS tool
tool=asiGetTool('spectreS) ; This is step 1.
; add the environment option myFile
asiAddEnvOption( tool       ; This is step 2.
    ?name                   'myFile
    ?type                   'fileName
    ?prompt                 "My File"
    ?value                  ""
    ?invalidateFunc         'asiInvalidateControlStmts
                            ; This is step 4. See the note
                            ; following this example.
)

; modify the step that sends control statements,to send myFile also.
flowchart = asiGetFlowchart( tool ) ; This is step 3.
asiChangeFlowchartStep( flowchart
    ?name                   'asiSendControlStmts
    ?postFunc               'mySendMyFile
)

; Write the routine (mySendMyFile) to send (the contents of) myFile to cdsSpice.
defmethod( mySendMyFile ( ( session spectreS_session ) )
; This is also step 3.
    let(( netlistDir customInclude customIncludeFile )

        ; Check if the option is set.
        if( nequal( asiGetEnvOptionVal( session 'myFile) "") then

        ; There is a mechanism in cdsSpice that allows you to
        ' create a file
        ; called: .customInclude in the netlist directory. If you
        ; then send
        ; a 'ptptop' command to cdsSpice, it includes the contents of
        ; .customInclude in the final netlist for your target
        ; simulator.

        ; Open the .customInclude file in the netlist directory.
        netlistDir = asiGetNetlistDir(session)
        when( rexMatchp( "Verilog" asiGetSimName( session ) )
                netlistDir = strcat( netlistDir "/analog" )
        )
        customInclude=strcat(netlistDir ".customInclude")
        customIncludeFile = outfile( customInclude "w")
```

```
        ; Add whatever you need to add to this file.
        ; In this case, perhaps it is the contents of 'myFile'.

        ; When you are finished, close the file.
        close( customIncludeFile )

        ; Send the 'ptprop' command to cdsSpice to include the
        ; contents of this file in the final netlist.
        asiSendSim( session "ptprop analog CustomIncludeFile 1" nil
        nil nil )
     else
        ; Send the command to turn off the inclusion of the file in
        ; the final netlist.
        asiSendSim( session "deprop analog CustomIncludeFile"  nil
        nil nil )
     )
     t
    )
)
```

Note: `asiInvalidateControlStmts` is a wrapper to `asiInvalidateFlowchartStep`, which invalidates the `asiSendControlStmts` step (as shown below):

```
defmethod( asiInvalidateControlStmts ( ( session spectreS_session ) )
asiInvalidateFlowchartStep( session 'asiSendControlStmts )
)
```

Changing Virtuoso Analog Design Environment Banner Menus

You can change the banner menus in the Simulation window.

1. Make a copy of the `simui.menus` file and make any modifications to the file that you require.

   The `simui.menus` file is located at

   `<`*`install_dir`*`>/tools/dfII/etc/tools/menus/simui.menus`

   The beginning of this file explains the syntax for menu definitions.

2. Place a copy of the file in the following location:

   `<`*`install_dir`*`>/tools/dfII/local/menus/simui.menus`

   This location is for site customization. You can also put `menus/simui.menus` in

   - The workArea (if there is one) or the current working directory
   - The projectArea (if TDM is in use)

- Your home directory

Note: You must create the `menus` directory to hold the *`simui.menus`* file.

3. Add the definition for the appropriate callback routine and associated code to your appropriate SKILL files.

Note: You can also add menus by using the SKILL *`hi`* calls.

Changing Banner Menus for a Particular Simulator

You can also customize the banner menus (in the Simulation window) for a particular simulator.

1. Modify the <*`simulator`*>.`menus` file.

   Look for the file in the following location.

   <*`install_dir`*>/tools/dfII/etc/tools/menus/<*`simulator`*>.menus

   If the file you need is here, make a copy of the file and make any modifications you want. If the file you need is not here, you can use the `spectreS` file (or another simulator file) as an example. Rename this file for your simulator and make any changes you want. (Do not include the angle brackets (<>) in the file name.)

   Note: The <*`simulator`*>.menus file can be for a Cadence simulator or a non-Cadence simulator.

2. Place a copy of the file in the following location:

   <*`install_dir`*>/tools/dfII/local/menus/<*`simulator`*>.menus

   This location is for site customization. You can also put `menus/`<*`simulator`*>`.menus` in

   - The workArea (if there is one) or the current working directory
   - The projectArea (if TDM is in use)
   - Your home directory

3. Add the appropriate callback routine and associated code to your appropriate SKILL files.

   Note: You can also add menus by using the SKILL *hi* calls.

Searching for SKILL Functions from the Finder

Use the Cadence SKILL API Finder tool to display the description and syntax of the Virtuoso Analog Design Environment SKILL functions. To open the SKILL API Finder

➡ From the CIW, select *Tools – SKILL Finder*.

The Cadence SKILL API Finder appears.

For information about using the SKILL API Finder, refer to the Using SKILL API Finder appendix in the *Cadence SKILL IDE User Guide* or click *Help – Finder Help* in the Finder.

2

Initialization Functions

This chapter describes the functions that let you initialize the simulation environment for your simulator and register your simulator.

asiInit<yourSimulator>

```
asiInit<yourSimulator>( o_tool )
     => t
```

Description

Calls the procedures to initialize your simulator's environment. This function must be defined for socket interfaces. Do not use it for direct interfaces.

You must write `asiInit<yourSimulator>`, where <*yourSimulator*> is the name of your simulator. Do not include the angle brackets (<>).

Arguments

o_tool	Simulation tool object.

Value Returned

`t`	Returns `t` when your simulator's environment is initialized. You must write `asiInit<yourSimulator>` to return `t`.

Example

```
procedure ( asiInitXYZ( tool )
    <insert your code>
    t
)
```

Creates the procedure that initializes the simulator environment for the XYZ simulator.

asiRegisterTool

```
asiRegisterTool(
     '<yourSimulator>
     [?class        s_className]
     [?private      s_private]
     [?initFunc     s_initFunc]
     [?mixedSig     s_mixedSig]
     )
     => t
```

Description

Registers your simulator and your initialization function.

Note: Replace <*yourSimulator*> with the name of your simulator. Do not include the angle brackets <>.

Arguments

<yourSimulator>
: Name of your simulator preceded by an apostrophe (').

s_className
: Specifies the class from which your simulator inherits information.
Default Value: 'analog.

s_private
: Optional argument that declares a simulator as "private," which means it does not appear in the list of simulators in the UI.
Valid Values: t indicates that the option does not appear in the list of simulators in the UI, nil indicates that the option appears in the list of simulators in the UI
Default Value: nil

Note: You can also use this argument to create a parent class from which other classes inherit information. Designers cannot see or directly use a parent class if *private* is set to t.

s_initFunc
: Initialization function for your simulator.
Default Value: asiInit<*yourSimulator*>, where <*yourSimulator*> is the name of your simulator.

s_mixedSig
: Set this argument to non-nil to specify a mixed-signal simulator.
Default Value: nil

Value Returned

`t`	Returns `t` when your simulator is registered.

Example

```
asiRegisterTool('XYZ)
```

Registers the XYZ simulator under the Analog Class.

asiInitDataAccessFunction

```
asiInitDataAccessFunction( o_tool )
     => t | nil
```

Description

Initializes the data access function for the tool. This function can be used by a third-party integrator to define their own data access functions.

Arguments

o_tool	Simulation tool object.

Value Returned

`t`	Returns `t` when your data access function is initialized.
`nil`	Returns `nil` when the function does not run successfully.

Example

```
(defmethod asiInitDataAccessFunction ( ( tool <yourSimulator> ) )
        asiDefineDataAccessFunction( tool 'VT '<yourSimulator>VT)
)
procedure( '<yourSimulator>VT(specifier dataDir simData)
        asiGetDrlData('tran specifier dataDir)
)
```

asiInitEnvOption

```
asiInitEnvOption( o_tool )
     => t | nil
```

Description

Initializes the tool-specific environment options for the tools that are derived from the asiAnalog class. This is not applicable for tools derived from the asiSocket class. This function can be used by third-party integrators to define their own environment options.

Arguments

o_tool	Simulation tool object.

Value Returned

`t`	Returns `t` when your tool-specific environment options are initialized.
`nil`	Returns `nil` when the function does not run successfully.

Example

```
defmethod( asiInitEnvOption ( ( tool <yourSimulator> ) )
    ;;; Initialize the environment options from the base class.
    callNextMethod()
    asiAddEnvOption( tool
        ?name        'MyOpt
        ?prompt      "Any String Option"
        ?value        "xyz"
        ?type        'string
    )
)
```

asiInitAnalysis

```
asiInitAnalysis( o_tool )
     => t | nil
```

Description

Initializes the tool-specific analysis options for the tools that are derived from the asiAnalog class. This is not applicable for tools derived from the asiSocket class. This function can be used by third-party integrators to define their own analysis options.

Arguments

o_tool	Simulation tool object.

Value Returned

t	Returns t when your tool-specific analysis options are initialized.
nil	Returns nil when the function does not run successfully.

Example

```
defmethod( asiInitAnalysis ( ( tool <yourSimulator> ) )
    asiAddAnalysis( tool
        ?name 'myAnal
        ?prompt "My Analysis"
    )
)
```

asiInitAdvAnalysis

```
asiInitAdvAnalysis( o_tool )
     => t | nil
```

Description

Initializes the tool-specific analysis options for the tools derived from the asiAnalog class. This method can be used by third-party integrators to define their own analysis options in ADE XL.

Arguments

o_tool	Simulation tool object

Values Returned

`t`	Returns `t` if tool-specific analysis options are initialized
`nil`	Returns `nil` if tool-specific analysis options are not initialized

Example

```
defmethod( asiInitAdvAnalysis (( tool <yourSimulator> ) )
    asiAddAnalysis( tool
        ?name 'myAdvancedAnalysis
        ?prompt "My Advanced Analysis"
    )
)
```

asiInitSimOption

```
asiInitSimOption( o_tool )
     => t | nil
```

Description

Initializes the simulation options for the tools that are derived from the asiAnalog class. This is not applicable for tools derived from the asiSocket class. This function can be used by third-party integrators to define their own simulation options.

Arguments

o_tool	Simulation tool object.

Value Returned

`t`	Returns `t` when your tool-specific simulation options are initialized.
`nil`	Returns `nil` when the function does not run successfully.

Example

```
(defmethod asiInitSimOption ( ( tool <yourSimulator> ) )
    asiAddSimOption( tool
        ?name 'mySimOption
        ?prompt "My Simulation Option"
    )
)
```

3

Netlisting Invocation Functions for Direct Integration

This chapter describes functions that let you invoke netlisting options for direct integration.

Overview of Standalone Invocation

The `si` flow supports all OSS-based backend netlisters and the ADE netlisters `spectre` and `Ultrasim`. The `si` flow does not work for any other simulator netlister integrated in ADE.

To use the `si` executable for the ADE netlisters, copy the `si.env` file into a new directory and execute the following command:

```
si -batch -cdslib <complete path to cds.lib>
```

This command generates the netlist as well as runs the simulation. Alternatively, use either the following command or the SKILL function `nl` to only generate the netlist:

```
si -batch -command nl
```

Important

The `-command netlist` option of the `si` command does not work for ADE netlisters and generates an error. Therefore, use the `-command nl` option as depicted above.

Note: The variable `nlFormatterClass` is used in addition to the usual OSS variables. It represents the symbolic name of the formatter class.

asiSetNetlistFormatterClass

```
asiSetNetlistFormatterClass(
     o_tool
     s_class )
     => s_class
```

Description

Registers the netlist formatting class with the tool. This function is normally called from the `asiInitFormatter` method and should be defined for the interface.

Arguments

o_tool	The object representing the simulator interface.
s_class	A symbol representing the formatter class for the simulator of interest.

Value Returned

s_class	A symbol representing the formatter class for the simulator of interest.

Example

```
asiSetNetlistFormatterClass( o_tool 'spectreFormatter) => spectreFormatter
```

asiCreateFormatter

```
asiCreateFormatter( g_session )
     => o_formatter
```

Description

First, a design object is created with the `nlCreateDesign` call, using the information on the OASIS session. Subsequently, the formatter is created with a call to `nlCreateFormatter`, using the information on the session. The formatter is added to the session and can be obtained with `asiGetFormatter`. This is a convenience routine that you cannot redefine, and the interface should not call it.

Arguments

o_session	The OASIS session object

Value Returned

o_formatter	The formatter object created.

Example

```
asiCreateFormatter( session )
```

asiCreateCdsenvFile

```
asiCreateCdsenvFile( s_toolName ) => t | nil
```

Description

Creates a .cdsenv file for the specified tool and dumps it to the current working directory. This is meant as a development utility for integrators only.

Arguments

s_toolName	The name of the tool. For example, spectre, spectreS, cdsSpice, hspiceS etc.

Value Returned

t	`.cdsenv` file created.
nil	No `.cdsenv` file created.

Example

```
asiCreateCdsenvFile('spectre )
```

Creates a file called `spectreCdsenvFile` containing spectre tool information in your current working directory.

```
asiCreateCdsenvFile('spectreS)
```

Creates `./spectreSCdsenvFile` for SpectreS tool.

asiGetFormatter

```
asiGetFormatter( o_session )
     => o_formatter | nil
```

Description

Returns the formatter created with the last `asiCreateFormatter` call. This is a convenience routine that you should not redefine and the interface should not call.

Arguments

o_session	The OASIS session object.

Value Returned

o_formatter	The formatter object.
`nil`	No formatter object is available for the session.

Example

```
asiGetFormatter( session )
```

asiGetSimInputFileName

```
asiGetSimInputFileName( o_session )
     => t_name
```

Description

Returns the name of the simulator input file. For the `asiAnalog_session` class, this is `input` followed by the return value of `asiGetSimInputFileSuffix`.

Arguments

o_session	The OASIS session object.

Value Returned

t_name	The name of the simulator input file.

Example

```
asiGetSimInputFileName( session ) => "input.ckt"
```

asiGetSimInputFileSuffix

```
asiGetSimInputFileSuffix( o_session )
```

Description

Returns the suffix used for the simulator input file. This method can be redefined, and must return a string, or a SKILL error will result.

Arguments

o_session	The OASIS session object.

Value Returned

t_name	The suffix of the simulator input file.

Example

```
asiGetSimInputFileSuffix( session ) => ".ckt"
```

4

Netlist Functions

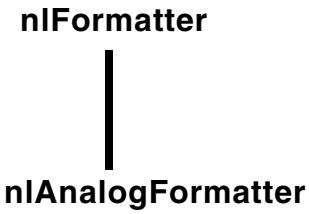
This chapter describes the following types of netlist functions:

The nlAnalogFormatter Class

The `nlAnalogFormatter` class represents the object that formats the design for the netlist file. It is responsible for printing the instances, the subcircuits, the comments, the global statements, and other information associated with the design connectivity.

The netlister calls several methods, such as `nlPrintInst`. The netlister traverses the design and calls the methods of the `nlAnalogFormatter` class. This class is tailored to the Spice 3 simulator and is derived from the more generic `nlFormatter` class:

Except as noted, the methods documented in this chapter may be redefined.



nlGetNetlister

```
nlGetNetlister(o_formatter)
o_netlister

OR

nlGetNetlister( o_instance )
=> o_netlister
```

Description

Returns an object representing the netlister.

Note: You can call this function, but do not redefine it.

Arguments

o_formatter	The formatter object.
o_instance	The instance object.

Value Returned

o_netlister	Returns the netlister object.

Example

```
o_netlister = nlGetNetlister(formatter)
nlGetNetlister( inst )
```

nlGetPCellParamSource

```
nlGetPCellParamSource( o_cellView l_parameters )
=> l_booleanValues / nil
```

Description

Identifies the source of value of the pcell parameters of the given cellview.

Arguments

o_cellview	cellView ID of the pcell.
l_parameters	List of parameters for which the source needs to be determined.

Value Returned

l_booleanValues	List of boolean values where each element of the list indicates whether corresponding parameter in input list is overridden on instances (for this pcell variant) or is taking a default value. The value t in this list indicates that the parameter is overridden on instance and nil indicates that the parameter has taken the default value.
nil	If there is an error, which can be because the cellview is called outside the context of netlister, or the cellView id is not a valid pcell.

Example

```
nlGetParamList (o_cellView)
l_parameters
nlGetPCellParamSource (o_cellView, l_parameters)
t nil nil t
```

nlGetToolName

```
nlGetToolName( o_formatter )
     => s_toolName
```

Description

Returns a symbol representing the simulator. It returns the value of the tool name. This name is used for the selection of the simulator information on the library component.

Note: You can call this function, but do not redefine it.

Arguments

o_formatter	The formatter object.

Value Returned

s_toolName	Returns the name of the tool.

Example

```
nlGetToolName( formatter )
```

nlInitialize

```
nlInitialize( o_formatter )
     => o_formatter | nil
```

Description

For the `nlFormatter` class, this method initializes the netlister. This method can be redefined for the simulator-specific netlister and is called by `nlCreateFormatter`. This method initializes all simulator-specific aspects of netlisting such as name mapping. For the `nlAnalogFormatter` class, this method sets a number of netlist options. These options and their values are shown in the table below. To inspect the value of an option, use `nlGetOption`.

For a description of all netlist options see "The Netlister Object" on page 99.

Option	value
`hierarchyDelimiter`	`"."`
`globalParamPrefix`	`"_gpar"`
`globalNetPrefix`	`""`
`instNamePrefix`	`"_inst"`
`invalidNetNames`	`'(("gnd!" "0"))`
inhModelName	t | nil specifies whether the simulator provides support for model name passing. A nil value would result in an error if model name is passed through the hierarchy for a stopping instance.
`linePrefix`	`"+"`
`linePostfix`	`nil`
`netNamePrefix`	`"_net"`
`mapInstFirstChar`	special characters and numbers
`mapInstInName`	all special characters
`mapModelFirstChar`	all special characters and numbers
`mapModelInName`	all special characters
`mapNetFirstChar`	all special characters and numbers
`mapNetInName`	special characters

Option	value
maxNameLength	1024
modulePrefix	"_sub"
paramNamePrefix	"_par"
subcktIndentString	four spaces
useInstNamePrefix	t

If you need to change any of these settings, use `callNextMethod()` to inherit as much as possible and then make the changes.

Arguments

o_formatter	The formatter object.

Value Returned

o_formatter	Returns the formatter object, if successful.
nil	Returns nil if the operation failed.

Example

```
defmethod( nlInitialize((formatter <yourSimulator>Formatter))
    let( (nlForm)
callNextMethod( formatter)
nlForm=nlGetNetlister( formatter )
nlSetOption( nlForm 'hierarchyDelimiter ":")
nlSetOption( nlForm 'maxNameLength 64)
formatter
) )
```

The above example uses `callNextMethod` to do the initialization in the `nlAnalogFormatter` class and then

- Changes the hierarchy delimiter from "." to ":"
- Changes the maximum name length from 1024 chars to 64 chars

nlPrintHeader

```
nlPrintHeader( o_formatter )
     => t | nil
```

Description

This method writes the beginning comment, adds `.GLOBAL`, and prints header comments.

Arguments

o_formatter	The formatter object.

Value Returned

`t`	Returns `t` when the comment is written.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintHeader ((formatter nlAnalogFormatter))
let( ((netlister nlGetNetlister(formatter)))
nlPrintString( netlister (nlGetOption netlister 'begComment))
nlPrintString( netlister "\n")
nlPrintString( netlister ".GLOBAL")
foreach( glob
    setof( x (nlGetGlobalNets netlister) (nequal x "gnd!"))
    nlPrintString(netlister " ")
    nlPrintString( netlister (nlMapGlobalNet netlister glob))
)
nlPrintHeaderComments( formatter)
nlPrintString(netlister "\n")
t
)
)
```

nlIncludeVerilogaFile

```
nlIncludeVerilogaFile( o_formatter t_filename t_master)
     => t | nil
```

Description

Prints the include statement for verilog-a cell views in the design. This is called before printing the footer for the netlist for all verilog-a modules. If your simulator does not support verilog-a views, call `nlError` in this method.

Note: The third argument (*t_master*) has been added from the 4.4.6 release onwards.

Arguments

o_formatter	The formatter object.
t_filename	The name of the file to include
t_master	The value for `-master` option on the include line

Value Returned

`t`	Success
`nil`	Failure

Example

```
(defmethod nlIncludeVerilogaFile ((obj <yourSimulator>Formatter) file master)
(when master
    (nlError (nlGetNetlister obj) "Cannot support multile modules with the same
name"))
  (nlPrintStringNoFold (nlGetNetlister obj) "ahdl_include \"" file "\"\n")
  )
```

nlIncludeVerilogFile

```
nlIncludeVerilogFile( o_formatter t_filename t_master)
=> t | nil
```

Description

Prints the include statement for verilog type text cell view in the design. This is called before printing the footer for the netlist for all verilog text cell views. If your simulator does not support verilog-a views, call `nlError` in this method.

Arguments

o_formatter	The formatter object.
t_filename	The name of the file to include
t_master	The value for `-master` option on the include line

Value Returned

`t`	Success
`nil`	Failure

Example

```
(defgeneric nlIncludeVerilogFile (obj_file_master))
```

nlIncludeDbDSPFTextFile

```
nlIncludeDbDSPFTextFile(
     o_formatter
     t_filename
     t_master
     )
     => t | nil
```

Description

Prints the dspf_include statement for DSPF type text cellview in the design.

Arguments

o_formatter	The OASIS session object.
t_filename	The name of the file to include in the design.
t_master	The value for `-master` option on the include line.

Value Returned

t	Returns `t` if the dspf_include statement is printed in the design.
`nil`	Returns `nil` otherwise.

Example

```
nlIncludeDbDSPFTextFile( formatter  cvFilePath master )
```

nlPrintFooter

```
nlPrintFooter( o_formatter )
     => t | nil
```

Description

This method is called at the end of netlisting. It does not print anything at the end of the netlist for the `nlAnalogFormatter` class.

Arguments

o_formatter	The formatter object.

Value Returned

`t`	Returns `t` if the footer is printed.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintFooter ((formatter xyzFormatter))
     callNextMethod()
     nlPrintComment( nlGetNetlister( xyzFormatter )
                 "End of netlist\n")
)
```

nlPrintSubcktHeaderComments

```
nlPrintSubcktHeaderComments(
     o_formatter
     o_cellView
     )
     => t
```

Description

Prints the comments for the subcircuit header and the mapping information when the `printSubcktComments` option is set.

Arguments

o_formatter	The formatter object.
o_cellView	The object representing the cell view.

Value Returned

`t`	Returns `t` in all cases.

Example

```
defmethod( nlPrintSubcktHeaderComments ((formatter
                                *xyz*Formatter) cv)
let( (nl)
     callNextMethod()
     nl = nlGetNetlister(formatter)
nlPrintComment( nl "anything else")
) )
```

nlPrintTopCellHeaderComments

```
nlPrintTopCellHeaderComments(
     o_formatter
     o_cellView
     )
     => t
```

Description

Calls `nlPrintSubcktHeaderComments` for the `nlAnalogFormatter` class.

Arguments

o_formatter	The formatter object.
o_cellView	The object representing the cell view.

Value Returned

`t`	Returns `t` in all cases.

Example

```
defmethod( nlPrintTopCellHeaderComments ((formatter
                                 xyzFormatter) cv)
nlPrintSubcktHeaderComments( formatter cv)
)
```

nlPrintTopCellFooterComments

```
nlPrintTopCellFooterComments(
     o_formatter
     o_cellView
     )
     => t
```

Description

Returns t at the `analogFormatter` level.

Arguments

o_formatter	The formatter object.
o_cellView	The object representing the cell view.

Value Returned

`t`	Returns `t` in all cases.

Example

```
defmethod( nlPrintTopCellFooterComments ((formatter
                                *xyz*Formatter) cv)
let( (nl)
     callNextMethod()
     nl = nlGetNetlister(formatter)
nlPrintComment( nl "anything else")
) )
```

nlPrintTopCellHeader

```
nlPrintTopCellHeader(
     o_formatter
     o_cellView
     )
     => t | nil
```

Description

Prints the header of the top-level circuit by calling `nlPrintTopCellHeaderComments`.

Arguments

o_formatter	The formatter object.
o_cellView	The object representing the cell view.

Value Returned

`t`	Returns `t` if the header is printed.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintTopCellHeader((formatter xyzformatter) cv)
       (nlPrintTopCellHeaderComments formatter cv)
)
```

nlPrintTopCellFooter

```
nlPrintTopCellFooter(
     o_formatter
     o_cellView
     )
     => t | nil
```

Description

Writes the top cell view footer. This function prints an empty line and calls `nlPrintTopCellFooterComments`.

Arguments

o_formatter	The formatter object.
o_cellView	The object representing the cellview.

Value Returned

`t`	Returns `t` if the footer is added.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintTopCellFooter ((formatter
                                  xyzFormatter) cv)
     callNextMethod()
     myTopCellFooter()
t)
```

nlPrintSubcktHeader

```
nlPrint SubcktHeader(
     o_formatter
     o_cellView
     )
     => t | nil
```

Description

Writes the header for a subcircuit following these steps: prints comments by calling `nlPrintSubcktHeaderComments`; prints the subckt begin keyword by calling `nlPrintSubcktBegin`; prints the subckt name by calling `nlPrintSubcktName`; prints the subckt terminal list by calling `nlPrintSubcktTerminalList.`

Arguments

o_formatter	The formatter object.
o_cellView	The object representing the cellview.

Value Returned

`t`	Returns `t` if the header for the subcircuit is added.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintSubcktHeader((formatter xyzformatter) cv )
let( (nl)
nl = nlGetNetlister(formatter)
    (nlPrintSubcktHeaderComments formatter cv)
    (nlPrintSubcktBegin formatter cv)
    (nlPrintSubcktName formatter cv)
    (nlPrintSubcktTerminalList formatter cv)
    (nlPrintString nl "\n")
) )
```

nlPrintSubcktFooter

```
nlPrintSubcktFooter(
     o_formatter
     o_cellView
     )
     => t | nil
```

Description

Writes the footer for the subcircuit. For the `nlAnalogFormatter` class, it prints `.ends`.

Arguments

o_formatter	The formatter object.
o_cellView	The object representing the cellview.

Value Returned

`t`	Returns `t` if the footer for the subcircuit is written.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintSubcktFooter ((formatter
                          xyzFormatter) cv)
nlPrintSubcktEnd( formatter cv)
nlPrintTopCellFooterComments( formatter cv)
t)
```

nlPrintSubcktFooterComments

```
nlPrintSubcktFooterComments(
     o_formatter
     o_cellView
     )
     => t
```

Description

Prints the comments for the subcircuit footer by printing the string '`End of subcircuit definition`.' preceded by the comment begin string.

Arguments

o_formatter	The formatter object.
o_cellView	The object representing the cellview.

Value Returned

`t`	Returns `t` if the subcircuit footer comments are printed.

Example

```
defmethod( nlPrintSubcktFooterComments ((formatter
                              xyzFormatter) cv)
let( (nl)
     callNextMethod()
     nl = nlGetNetlister(formatter)
nlPrintComment( nl "anything else")
) )
```

nlPrintInstComments

```
nlPrintInstComments(
     o_formatter
     o_instance
     )
     => t
```

Description

Prints the comments for an instance.

Arguments

o_formatter	The formatter object.
o_instance	The object representing the instance.

Value Returned

t	Returns t if the comments for the instance are printed.

Example

```
defmethod( nlPrintInstComments ((formatter
                          xyzFormatter) inst)
let( (nl)
     callNextMethod()
     nl = nlGetNetlister(formatter)
nlPrintComment( nl "anything else")
) )
```

nlPrintInst

```
nlPrintInst(
     o_formatter
     o_instance
     )
     => t | nil
```

Description

Prints the netlist statement for an instance. This is the default netlist procedure for a component.

It prints the following:

- instance comments by calling `nlPrintInstComments`
- a string to indent (for the top-level circuit, this is an empty string; for subcircuits, this is the value of the `subcktIndentString` netlist option.)
- instance name
- signals
- model name
- instance parameters in name=value pairs.

Note: If a netlist procedure is specified as a `netlistProcedure` in the CDF simInfo, that procedure is called instead of `nlPrintInst`.

Note: The end of the instance line, `\n`, is not printed by this method or the netlist procedure. The end of the line is printed by the `nlPrintInstEnd` method.

Arguments

o_formatter	The formatter object.
o_instance	The object representing the instance.

Value Returned

`t`	Returns `t` if the netlist statement is printed.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintInst ((formatter nlAnalogFormatter) inst)
nlPrintInstComments( formatter inst)
nlPrintIndentString( nlGetNetlister( formatter ))
nlPrintInstName( formatter inst)
nlPrintInstSignals( formatter inst)
nlPrintModelName( formatter inst)
nlPrintInstParameters( formatter inst)
t
)
```

nlPrintInstEnd

```
nlPrintInstEnd(
     o_formatter
     o_instance
     )
     => t | nil
```

Description

Prints the end of the instance statement, which is a return (`\n`). This method is called by the netlister after the netlist procedure or by `nlPrintInst`.

Arguments

o_formatter	The formatter object.
o_instance	The object representing the instance.

Value Returned

`t`	Returns `t` if the end of the instance statement is printed.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintInstEnd ((formatter nlAnalogFormatter) inst)
    nlPrintString( nlGetNetlister( formatter) "\n")
    t
)
```

nlPrintSubcktBegin

```
nlPrintSubcktBegin(
     o_formatter
     o_cellView
     )
     => t | nil
```

Description

Prints the `.subckt` keyword for the `nlAnalogFormatter` class. This method is called by `nlSubcktHeader`.

Arguments

o_formatter	The formatter object.
o_cellView	The object representing the cellview.

Value Returned

`t`	Returns `t` if the keyword is printed.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintSubcktBegin ((formatter nlAnalogFormatter) _cv)
    nlPrintString( nlGetNetlister( formatter) ".subckt")
    t
)
```

nlPrintSubcktName

```
nlPrintSubcktName(
     o_formatter
     o_cellView
     )
     => t | nil
```

Description

Prints a space and the simulator name of the subcircuit. This method is used by `nlSubcktHeader`.

Arguments

o_formatter	The formatter object.
o_cellView	The object representing the cellview.

Value Returned

`t`	Returns t if the simulator name is printed.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintSubcktName ((formatter nlAnalogFormatter) cv)
    let( ( (nl nlGetNetlister(formatter))
    nlPrintString( nl " ")
    nlPrintString( nl nlGetSimName (cv) )
    t ) )
```

nlPrintSubcktEnd

```
nlPrintSubcktEnd(
     o_formatter
     o_cellView
     )
     => t | nil
```

Description

For the `nlAnalogFormatter` class, prints the `.ends` keyword, followed by a space and the simulator name of the subcircuit, to mark the end of the subcircuit definition. It is called by `nlPrintSubcktFooter`.

Arguments

o_formatter	The formatter object.
o_cellView	The object representing the cellview.

Value Returned

`t`	Returns `t` if the keyword followed by a space and the simulator name is printed.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintSubcktEnd ((formatter nlAnalogFormatter) cv)
    let( ( (nl nlGetNetlister(formatter))
    nlPrintString( nl ".ends")
    nlPrintString( nl nlGetSimName (cv) )
    nlPrintString( nl "\n")
    t
) )
```

nlPrintHeaderComments

```
nlPrintHeaderComments( o_formatter )
     => t | nil
```

Description

Prints the comments for the netlist file, including the library, cell, and view names of the top-level cellview of the design netlisted. This method only has effect when the `printFileComments` netlist option is set.

Arguments

o_formatter	The formatter object.

Value Returned

`t`	Returns `t` if the comments are printed.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintHeaderComments ((formatter
                                   xyzFormatter) inst)
let( (nl)
nl = nlGetNetlister(formatter)
     callNextMethod()
     nl = nlGetNetlister(formatter)
     nlPrintComment( nl "anything else")
) )
```

nlPrintSubcktParameters

```
nlPrintSubcktParameters(
     o_formatter
     o_cellView
     )
     => t | nil
```

Description

Prints the passed parameters for the subcircuit definition.

Arguments

o_formatter	The formatter object.
o_cellView	The object representing the cellview.

Value Returned

`t`	Returns `t` if the parameters are printed.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintSubcktParameters
                 ((formatter nlAnalogFormatter) cv)
let( ((plist nlGetParamList( cv))
                    (nl nlGetNetlister( formatter)))
when( plist
      nlPrintString( nl ".PARAMETERS")
           foreach( param plist
               cond(
                    ((cadr param) nlPrintString( nl " "
                              (car param) "=" (cadr param)))
                     (t nlPrintString( nl " " (car param)))
                   )
          )
      nlPrintString( nl "\n")
)))
```

nlPrintSubcktTerminalList

```
nlPrintSubcktTerminalList(
     o_formatter
     o_cellView
     )
     => t | nil
```

Description

Prints the simulator names of the signals connected to the terminals for a subcircuit definition and handles the signals resulting from inherited connections at a lower level .

Arguments

o_formatter	The formatter object.
o_cellView	The object representing the cellview.

Value Returned

`t`	Returns `t` if the simulator names are printed.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintSubcktTerminalList ((formatter
                              xyzFormatter) cv )
     let( ( (nl nlGetNetlister(formatter))
     foreach( net nlGetSimTerminalNets( cv )
          nlPrintString( nl " " )
     nlPrintString( nl net ) )
) )
```

nlPrintInstName

```
nlPrintInstName(
     o_formatter
     o_instance
     )
     => t | nil
```

Description

Prints the simulator name of the instance, taking the instance name prefix specified on the component into account when the simulator so requires. This is determined with the `useInstNamePrefix` netlist option.

Arguments

o_formatter	The formatter object.
o_instance	The object representing the instance.

Value Returned

`t`	Returns `t` if the simulator name is printed.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintInstName ((formatter
                              xyzFormatter) inst)
nlPrintString( nlGetNetlister( formatter)
                        nlGetSimName (inst ) ) )
```

nlPrintInstSignals

```
nlPrintInstSignals(
     o_formatter
     o_instance
     )
     => t | nil
```

Description

Prints the simulator names of the signals according to the terminal order specified on the component, using the `nlGetSignalList` method of the instance.

Arguments

o_formatter	The formatter object.
o_instance	The object representing the instance.

Value Returned

`t`	Returns `t` if the simulator names of the signals are printed.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintInstSignals ((formatter
                                xyzFormatter) inst)
let( (sigs (fp nlGetNetlister(formatter)) )
      ;; print terminals
      when( sigs = (nlGetSignalList inst)
           foreach( sig sigs
                nlPrintString( fp sig " ")
           ))))
```

nlPrintModelName

```
nlPrintModelName(
     o_formatter
     o_instance
     )
     => t | nil
```

Description

Prints the model name. The `nlGetModelName` for the instance is used for the model name.

Arguments

o_formatter	The formatter object.
o_instance	The object representing the instance.

Value Returned

`t`	Returns `t` if the model name is printed.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintModelName ((formatter xyzFormatter) inst)
     nlPrintString( nlGetNetlister( formatter ) " "
               nlGetModelName( inst ))
)
```

Note: If your simulator supports model name passing, you can use the function `nlIsModelNameInherited` and alter the formatting of the instance line for the scenario as per the requirements of your simulator.

nlPrintInstParameters

```
nlPrintInstParameters(
     o_formatter
     o_instance
     )
     => t | nil
```

Description

Prints the instance parameters in name=value pairs.

Arguments

o_formatter	The formatter object.
o_instance	The object representing the instance.

Value Returned

`t`	Returns `t` if the instance parameters are printed.
`nil`	Returns `nil` if there is an error.

Example

```
defmethod( nlPrintInstParameters
                      ((formatter nlAnalogFormatter) inst)
let( (val (nl nlGetNetlister( inst) ) )
;; print properties
foreach( par nlGetParamList( inst)
     unless(
          val = nlGetParamStringValue( inst par)
          nlPrintString( nl)
                              " " get_string (par) "=" val)
      ))))
```

The Netlister Object

The netlister object represents the engine that produces the netlist. This object drives the format object and provides all necessary services such as name mapping. The netlister is the incremental, hierarchical netlister. You can obtain the netlister object from the formatter using `nlGetNetlister`.

None of the methods defined for the netlister can be redefined.

The options that can be set with the `nlSetOption` method are given in the table below. The values for the `nlAnalogFormatter` are provided in "nlInitialize" on page 70.

This object represents the incremental hierarchical netlister. The netlist options are set with nlSetOption. This section also lists the netlist options.

Netlist Options

A number of options have been added for direct simulation.

Note: Call `nlGetOptionNameList` to get a complete list of netlist options.

begComment

Comment string for lines that begin with a comment for your simulator.

constantsList

A list of names which are used as constant names by your simulator. These names are mapped by the netlister when used as passed parameter names.

globalNetPrefix

Specifies the string the formatter should use as the prefix for all global nets in a netlist.

globalParamPrefix

Specifies the string the formatter should use as the prefix for all global parameters in a netlist.

hierarchyDelimiter

Character used by the simulator for delimiting levels of hierarchy when it flattens the hierarchical netlist. Used for mapping result names.

instNameDifferentFrom

A list that contains the name types that the instance names cannot collide with. The types can be one or more of "model", "net", and "parameter".

instNamePrefix

Specifies the name prefix to be used when the netlister generates a unique instance name.

instTermDelimiter

The delimiter between the instance and the terminal used by the simulator.

invalidModelNames

A list of names which are invalid as model names for your simulator.

invalidInstNames

A list of names which are invalid as instance names for your simulator.

invalidNetNames

A list of names which are invalid as net names in the target tool net name space.

invalidParamNames

A list of names which are invalid as parameter names for your simulator.

linePostfix

String placed at the end of the current line when a line output to the netlist exceeds the maximum line length of the simulator and must be split into two lines. This variable is used by the `nlPrintString` function.

linePrefix

String placed at the beginning of the next line when a line output to the netlist exceeds the maximum line length of the simulator and must be split into two lines. This variable is used by the `nlPrintString` method.

mapInstFirstChar

A SKILL list that specifies which characters may not begin an instance name for the target simulator. 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 instance names.For example, if the value of this option is

```
'( "." )
```

then the name ".`inst24`" is mapped to a name that is generated by the netlister, starting with the value of the `instNamePrefix` option. For the `nlAnalogFormatter` class the name could be "`_inst3`".

mapInstInName

A SKILL list that specifies which characters may not be contained in an instance name for the target simulator. 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 instance names.

For example, if the value of this option is

```
'( "." ("<" "_") (">") )
```

then the name "inst.24" is mapped to a name that is generated by the netlister, starting with the value of the instNamePrefix option. For the nlAnalogFormatter class the name could be "_inst3". The name "inst<1>" could be mapped to "inst5_1".

mapModelFirstChar

A SKILL list that specifies which characters may not begin a subcircuit name for the target simulator. 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 subcircuit names. For example, if the value of this option is

```
'( "." )
```

then the name ".sub24" is mapped to a name that is generated by the netlister, starting with the value of the modulePrefix option. For the nlAnalogFormatter class the name could be "_sub3".

mapModelInName

A SKILL list that specifies which characters may not be contained in an subcircuit name for the target simulator. 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 subcircuit names.

For example, if the value of this option is

```
'( "." )
```

then the name "sub.24" is mapped to a name that is generated by the netlister, starting with the value of the modulePrefix option. For the nlAnalogFormatter class the name could be "_sub3".

mapNetFirstChar

A SKILL list that specifies which characters may not begin a name for the target simulator. 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 net names. For example, if the value of this option is

```
'( "." )
```

then the name ".net24" is mapped to a name that is generated by the netlister, starting with the value of the netNamePrefix option. For the nlAnalogFormatter class the name could be "_net3".

mapNetInName

A SKILL list that specifies which characters may not be contained in a name for the target simulator. 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 net names. For example, if the value of this option is

```
'( "." ("<" "_") (">") )
```

then the name "net.24" is mapped to a name that is generated by the netlister, starting with the value of the netNamePrefix option. For the nlAnalogFormatter class the name could be "_net3". The name "net5<1>" could be mapped to "net5_1"

mapSubcktTermsToOrder

When set, the subcircuit terminal names are mapped to the order of the terminal.

maxLineLength

Maximum line length of a line output to the netlist file when using the `nlPrintString` function. When the maximum is reached, the line is broken at the first blank space before this limit is and continued on the next line. If there is no blank space on the line, the line is broke at the limit and continued on the next line.

maxNameLength

Maximum number of character allowed in a simulator name. Used for mapping.

modelNameDifferentFrom

A list that contains the name types that the subcircuit names cannot collide with. The types can be one or more of "instance", "net", and "parameter".

modulePrefix

Specifies the string the formatter uses as the prefix for a mapped subcircuit name in a netlist.

netNameDifferentFrom

A list that contains the name types that the net names cannot collide with. The types can be one or more of "model", "instance", and "parameter".

netNamePrefix

Specifies the name prefix to be used when the netlister generates a unique signal name.

paramNameDifferentFrom

A list that contains the name types that the parameter names cannot collide with. The types can be one or more of "model", "net", and "instance".

paramNamePrefix

Specifies the name prefix to be used when the netlister generates a unique parameter name.

printFileComments (t)

Print the standard comments at the beginning and end of the netlist file.

printInstComments(nil)

Print the standard comment before each instance line.

printSubcktComments (t)

Print the standard comments at the beginning and end of each subcircuit, including the top-level circuit

printSubcktTerminalComments(nil)

Print the mapping of the subckt terminals to the beginning of the subcircuit

softLineLength

The desirable maximum line length of a line output to the netlist file when using the `nlPrintString` function. When the maximum is reached, the line is broken at first blank space before this limit, and continued on the next line. If no space is available, the output is continued on the same line until a space is encountered, or until `maxLineLength` is reached.

subcktIndentString

A string that represents the indentation in a subcircuit.

subcktInstPrefix

If `useInstNamePrefix` is set, and no prefix is specified on the CDF, then this prefix is used for subcircuit instances.

suffixMap

A SKILL list which specifies the suffixes that need to be mapped. Each element in the list could be a symbol or a string or another list which has either one or two elements. For example, if the value of this option is

```
'( '( "M" "Meg" ) '( 'Y ) 'm)
```

then "M" must be mapped to "Meg", and "Y" and "m" must be mapped to the internal default.

useInstNamePrefix (t)

When set, the name prefix is taken into account for instance names.

nlError

```
nlError( o_netlister t_error )
     => nil
```

Description

Issues a user error. The error is printed immediately and it is collected on the object. In this way, if netlisting is interrupted, the user is aware of any errors that occurred during netlisting. All errors are printed to the netlist log file.

Arguments

o_netlister	The netlister object.
t_error	The description of the error in the form of a format string.

Value Returned

`nil`	Returns `nil` in all cases.

Example

```
nlError(nl sprintf("Missing Parameter ‘%s’ in %s" param
    nlGetSimName(inst) ) )
```

nlObjError

```
nlObjError(
     o_netlister
     o_object
     t_error
     )
     => nil
```

Description

Similar to nlError, but prints a description of the object along with the error message. The description includes the library name, the cell name, the view name, and the instance name in case the object is an instance.

Arguments

o_netlister	The netlister object.
o_object	The instance or the cellview object.
t_error	The description of the error in the form of a format string.

Value Returned

`nil`	Returns `nil` in all cases.

Example

```
nlObjError(nl inst sprintf("Missing Parameter '%s' in" param ) )
```

nlGetDesign

```
nlGetDesign( o_netlister )
     => o_design
```

Description

Returns the design object.

Arguments

o_netlister	The netlister object.

Value Returned

o_design	Returns the design object.

Example

```
nlGetDesign( netlister )
```

nlGetGlobalNets

```
nlGetGlobalNets( o_netlister )
     => l_globalNets
```

Description

Returns the list of global nets. This method should only be used in the `nlPrintHeader` method of the formatter.

Globals may be included that are not used, due to inherited connections.

Arguments

o_netlister	The netlister object.

Value Returned

l_globalNets	Returns the list of global signals.

Example

```
nlGetGlobalNets( netlister )
```

nlGetNetlistDir

```
nlGetNetlistDir( o_netlister )
     => t_netlistDir
```

Description

Returns the netlist directory.

Arguments

o_netlister	The netlister object.

Value Returned

t_netlistDir	Returns the netlist directory.

Example

```
nlGetNetlistDir( netlister )
```

nlDisplayOption

```
nlDisplayOption( o_netlister )
     => t
```

Description

Prints the option names available on this object along with their values.

Arguments

o_netlister	The netlister object.

Value Returned

t	Returns `t` in all cases.

Example

```
nlDisplayOption( netlister )
```

nlGetCurrentSwitchMaster

```
nlGetCurrentSwitchMaster( o_netlister )
     => g_id
```

Description

This function returns the database ID for the current switch master for the instance in hierarchical incremental netlisting . This function should only be used while printing the instance statement. Avoid using this function if possible.

Arguments

o_netlister	The netlister object

Value Returned

g_id	The Database ID for the current switch master

Example

```
nlGetCurrentSwitchMaster( netlister )
```

nlGetOption

```
nlGetOption(
     o_netlister
     s_name
     )
     => g_value
```

Description

Returns the value of the option.

Arguments

o_netlister	The netlister object.
s_name	The name of the netlist option.

Value Returned

g_value	Returns the value of the netlist option.

Example

```
nlGetOption( netlister 'begComment)
```

nlGetOptionNameList

```
nlGetOptionNameList( o_netlister )
     => l_names
```

Description

Returns the list of option names available on this object.

Arguments

o_netlister	The netlister object.

Value Returned

l_names	Returns the list of option names.

Example

```
nlGetOptionNameList( netlister )
```

nlMapGlobalNet

```
nlMapGlobalNet(
     o_netlister
     t_net
     )
     => t_map
```

Description

Maps a global net (signal) to the simulator name. This should only be used in the `nlPrintHeader` method of the formatter. Use at any other time is an error.

Arguments

o_netlister	The netlister object.
t_net	The schematic name of the global net.

Value Returned

t_map	Returns the mapped name of the global net.

Example

```
nlMapGlobalNet( netlister globalNet )
```

nlInfo

```
nlInfo(
     o_netlister
     t_info
     [ g_arg ... ]
     )
     => t
```

Description

Sends an informational message to the calling application.

Arguments

o_netlister	The netlister object.
t_info	The message description, a format string.
g_arg	Arguments used for printing with the format string.

Value Returned

t	Returns t in all cases.

Example

```
nlInfo(nl "My info message.\n")
```

nlSetOption

```
nlSetOption(
     o_netlister
     s_option
     g_value
     )
     => t | nil
```

Description

Sets an option value. For information about options, see Netlist Options on page 4-41.

Note: This function can only be called during initialization. If it is called during netlisting, an error results.

Arguments

o_netlister	The netlister object.
s_name	The name of the option.
g_value	The value to which the option is set. The value type must be appropriate for the option name.

Value Returned

`t`	Returns `t` if the option value is set.
`nil`	Returns `nil` if there is an error.

Example

```
nlSetOption( netlister ‘maxNameLength 1024)
nlSetOption( netlister ‘hierarchyDelimiter ".")
```

nlWarning

```
nlWarning(
     o_netlister
     t_warning
     [ g_arg ... ]
     )
     => t
```

Description

Issue a warning to the user.

Arguments

o_netlister	The netlister object.
t_warning	The description of the warning message in the form of a format string.
g_arg	Arguments used for printing with the format string.

Value Returned

t	Returns t in all cases.

Example

```
nlWarning(nl "My warning message.\n")
```

nlPrintComment

```
nlPrintComment(
     o_netlister
     [t_arg1 t_arg2 ...]
     )
     => t | nil
```

Description

Prints a comment. Use this method to print all comments. This method uses the comment character and line wrapping. A subsequent `nlPrintf` call inserts the end comment string.

Arguments

o_netlister	The netlister object.
t_arg1, t_arg2	Arguments to formatted output.

Value Returned

`t`	Returns `t` if the comments are printed.
`nil`	Returns `nil` if there is an error.

Example

```
nlPrintComment( netlister "My comment" )
```

nlPrintIndentString

```
nlPrintIndentString( o_netlister )
     => t | nil
```

Description

Prints the indent string for the instance statement. When inside the top-level circuit, the empty string is printed. When inside subcircuit definitions, the value of the `subcktIndentString` netlist option is printed.

Arguments

o_netlister	The netlister object.

Value Returned

`t`	Returns `t` if the indent string is printed.
`nil`	Returns `nil` if there is an error.

Example

```
nlPrintIndentString( netlister )
```

nlPrintString

```
nlPrintString( o_netlister, @rest t_args)
     => t | nil
```

Description

Prints the string arguments to the file. This function does the required line folding, prefixing, and, postfixing.

Arguments

o_netlister	The netlister object.
t_args	The strings to be printed.

Value Returned

`t`	Returns `t` if the string argument is printed.
`nil`	Returns `nil` if there is an error.

Example

```
nlPrintString( netlister, "my string\n" )
```

nlPrintStringNoFold

```
nlPrintStringNoFold( o_netlister, @rest t_args)
     => t | nil
```

Description

Prints the string arguments to the file, like the function `nlPrintString`. This function does the required prefixing and postfixing, but does not fold the line until a newline character or `maxLineLength` is encountered. While in a 'no fold' print mode, calls cannot be made to nlPrintString.

Arguments

o_netlister	The netlister object.
t_args	The strings to be printed.

Value Returned

`t`	Returns `t` if the string argument is printed.
`nil`	Returns `nil` if there is an error.

Example

```
nlPrintStringNoFold( netlister, "my string\n" )
```

Methods for Instances

During netlisting, instances are represented by objects. These objects cannot be redefined by the simulator interface.

Important

Do not redefine any of the methods in this section.

nlIsModelNameInherited

```
nlIsModelNameInherited( o_instance )
     => t | nil
```

Description

Returns `t` if the model name for the stopping instance is passed through the hierarchy through parameters.

Arguments

o_instance	The instance object.

Value Returned

`t`	The model name will be passed from a higher level.
`nil`	The model name is specified at the instance.

Example

```
nlIsModelNameInherited( inst )
```

nlGetFormatter

```
nlGetFormatter( o_instance )
     => o_formatter
```

Description

Returns the formatter.

Arguments

o_instance	The instance object.

Value Returned

o_formatter	Returns the formatter object used for netlisting.

Example

```
nlGetFormatter( inst )
```

nlGetSimName

```
nlGetSimName( o_instance )
t_name

OR

nlGetSimName( o_cellView )
=> t_simName
```

Description

If the input type is cellview object, then returns the simulator name of the subcircuit. If the input type is instance object, then returns the mapped name of the instance. The name returned depends on the `useInstNamePrefix` netlist option. When it is not set, the name prefix is not taken into account. However, for instances representing interface elements, the name prefix is always taken into account. No mapping is performed for interface elements.

For instance object, when the `useInstNamePrefix` netlist option is set, this method takes the name prefix into account. The prefix of a subcircuit does not have to be specified on the CDF of each subcircuit. If the prefix is specified on the CDF, it is used. Otherwise, the subcktInstPrefix netlist option is used.

Arguments

o_instance	The instance object.
o_cellView	The cellview object.

Value Returned

t_name	Returns the mapped instance name if the input type is instance object.
t_simName	Returns the simulator name if the input type is cellview object.

Example

```
nlGetSimName( inst )
nlGetSimName( cv )
```

nlGetSignalList

```
nlGetSignalList( o_instance )
     => l_signals
```

Description

Returns the list of mapped signal names for the instance according to the terminal order specified for the cellview. Use this method for printing instances to the netlist by the `nlPrintInst` method of the formatter. The terminal order for schematic subcircuits is determined by the pin order property on the schematic, or by the termOrder property on the CDF, or by the system, in that order.

Note: Ordering requirements, such as the Verilog® requirement that outputs occur before inputs, are not satisfied in this release.

Signal buses are handled in scalar form. For example, `net10<0:3>` is mapped as `net10_0`, `net10_1`, `net10_2`, and `net10_3`.

Arguments

o_instance	The instance object.

Value Returned

l_signals	Returns the list of mapped signals for the instance.

Example

```
nlGetSignalList( inst )
```

nlGetTerminalList

```
nlGetTerminalList( o_instance )
     => l_terminals
```

Description

Returns the list of terminal names in the order specified on the pin order property on the schematic, or on the termOrder property on the CDF, or on the cellview of the instance, in that order. This method should not be used by the formatter. In contrast to `nlGetSignalList`, buses are not handled individually: a terminal such as `out<0:3>` is represented in its original form.

Arguments

o_instance	The instance object.

Value Returned

l_terminals	Returns the list of terminals of the instance.

Example

```
nlGetTerminalList( inst )
```

nlGetTerminalSignalName

```
nlGetTerminalSignalName(
     o_instance
     t_terminal
     [x_bit]
     )
     => t_signal
```

Description

Returns the name of the signal connected to the terminal.

Arguments

o_instance	The instance object.
t_terminal	The schematic name of the terminal.
x_bit	A non-negative integer value representing the bit number. The default is `0`.

Value Returned

t_signal	Returns the simulator name of the signal.

Example

```
nlGetTerminalSignalName( inst "in" )
```

nlGetNumberOfBits

```
nlGetNumberOfBits(
     o_instance
     t_terminal
     )
     => x_bits
```

Description

Returns the number of bits on the instance and terminal specified.

Arguments

o_instance	The instance object.
t_terminal	The schematic name of the terminal.

Value Returned

x_bits	Returns a non-negative integer value representing the bit number.

Example

```
nlGetNumberOfBits( inst "in")
```

nlGetModelName

```
nlGetModelName( o_instance )
     => t_modelName
```

Description

This method must be used to obtain the model name of an instance. Use of this method assures consistency in netlisting across interfaces. For instances that represent subcircuits, this method returns a name chosen by the netlister, using the `modulePrefix` netlist option.

The method `nlGetModelName` returns the following value:

1. The value of the model parameter on the instance, if this instance has CDF and if this parameter has a value.
2. The value of the `modelName` parameter on the instance if this instance has no CDF and if this parameter has a value.
3. The `componentName` entry on the view-specific information or on the simulator section on the `simInfo` section of the instance CDF if this has a value.
4. The name of the cell.

These rules have consequences for the `simInfo` section for cells of which the views are used as stop cell-views. A number of simulators such as Spectre require a component or model name to define the type of component. An example of a component name is `resistor`, and an example of a model name is `nmos101`. This method addresses both model names and component names. For these simulators the `useInstNamePrefix` option is set to `nil`. For the "`spectre`" interface the `analogLib` `res` cell has a `componentName` entry set to "`resistor`". When the model name is not specified on an instance of a `res` cell, this value is used.

Arguments

o_instance	The instance object.

Value Returned

t_modelName	Returns the name of the model.

Example

```
nlGetModelName( inst )
```

nlGetParamList

```
nlGetParamList( o_instance )
l_parameters

OR

nlGetParamList( o_cellView )
=> l_parameters
```

Description

Returns the list of parameters of the specified instance or cellview.

These parameters have been collected while the netlist was generated. In other words, all `pPar` and `atPar` expressions are collected. In addition, any `atPar` expressions from lower levels that are not resolved in the cellview are collected. Any interdependence of default values is handled by ordering the parameters so that those with independent default values are first.

Arguments

o_instance	The instance object.
o_cellView	The cellview object.

Value Returned

l_parameters	Returns the list of parameters. Each element is a symbol, representing the name of the parameter.

Example

```
nlGetParamList( inst )
nlGetParamList( cv )
```

nlGetParamStringValue

```
nlGetParamStringValue(
     o_inst
     s_parameter
     )
     => t_value | nil
```

Description

Returns a string representing the parameter value for the instance and parameter name.

The parameter name is the simulator name for the parameter. Values returned are strings irrespective of their original type. If the parameter does not exist or if the value is a blank string, `nil` is returned.

Note that `none` is no longer recognized as a special keyword. Previously it was treated as being equivalent to `nil`, but now it is treated as a design variable.

The parameters are evaluated according to the common evaluator. For details, see Chapter 9 of the Compatibility Guide. This means that values of CDF parameters can reference parameters with `iPar` or `pPar` that have NLP expressions. The CDF default value can be overridden with an instance property. The property type of that property must be a string.

The following table shows a synopsis of that scheme:

This property...	Is CDF?	And evaluates to...
NLP	yes	nil
NLP	no	As NLP
string	yes	As specified on the CDF
string	no	The literal string

If the parameter is a CDF parameter, this routine takes all aspects into account, including the CDF parameter type and the `parseAsCEL` attribute.

This procedure handles all look-up for `pPar`, `iPar`, and `atPar`, as well as the parameter name map specified for the component.

The AEL `pPar` and the NLP "`+`" calls are replaced by the enclosed variable. The passed parameter found is automatically collected. The netlister tracks these parameters for appropriate invalidation purposes.

The AEL `iPar` and the NLP "`~`" calls are replaced by the appropriate property values.

The use of the AEL `atPar` function and the NLP "`@`" calls is discouraged. Like `pPar`, these are replaced with the variable specified, and the parameter specified is declared as a passed parameter in the subcircuit.

Important

The AEL `dotPar` function and the NLP "`.`" are not supported, and a netlist error is reported if they are used.

The suffixes are substituted according to the suffix mapping specified. Thus, a schematic value "`1n`" may result in "`1e-9`".

Note: Only a limited set of NLP expressions is translated appropriately. The format accepted is limited to:

```
[<operator> <parameter name>],
[<operator> <parameter name>: % ],
[<operator> <parameter name>: % : <default value>],
```

with the operator being `@`, `+`, or `~`.

Many NLP expressions are therefore not supported:

- Complex formatting such as `:abc%d%e`
- Formatting on the default clause
- Modifiers in front of the operator, such as time and capacitance scaling
- Nesting in the default value or value formatting section

Note: For information about `iPar`, `pPar`, `atPar` and `dotPar` functions, refer to the Passing Parameters in a Design section in the Component Description Format user guide.

Arguments

o_instance	The instance object.
s_parameter	The name of the parameter.

Value Returned

t_value	The string form of the parameter value.
nil	The parameter was not found, the value was `nil`, the value was an empty string, or the value was a string consisting of only white space.

Example

```
nlGetParamStringValue( inst 'tvpairs )
```

nlGetId

```
nlGetId( o_cellView )
g_id

OR

nlGetId( o_instance)
=> t_id
```

Description

Returns the database ID for the instance. If the instance represents a cdba instance, this id is a database ID. Avoid using this function if possible, and use type-checking precautions.

Arguments

o_cellView	The cellview object.
o_instance	The instance object.

Value Returned

t_id	Returns the database ID of the instance or the cellview object.

Example

```
nlGetId( inst )
nlGetId( cv )
```

nlIncludeSrcFile

```
nlIncludeSrcFile(
     o_formatter
     t_filename
     )
     => t|nil
```

Description

Prints the include statement for instances bound to source files using Hierarchy Editor. This is called while printing the footer for the netlist for all the source files to which any instance in the design is bound. If your simulator does not support source file bindings, call `nlError` in this method.

Arguments

`o_formatter`	The formatter object.
`t_filename`	The name of the source file bound to an instance.

Value Returned

t	Returns `t` if the include statement is printed.
nil	Returns `nil` if the operation failed.

Example

```
(defmethod nlIncludeSrcFile ((obj <yourSimulator>Formatter) file)
(let ((nl (nlGetNetlister obj)))
(nlPrintStringNoFold nl "include '" simplifyFilename(file) "'")
(nlPrintStringNoFold nl "\n")
))
```

nlPrintComments

```
nlPrintComments( o_instance )
     => t
```

Description

Prints the comments for the instance being netlisted.

Arguments

o_instance	The instance object.

Value Returned

t	Returns t if the comments for the instance are printed.

Example

```
nlPrintComments( inst )
```

hnlGetInstanceCount

`hnlGetInstanceCount()`

Description

Returns the number of instances in the design most recently netlisted in the same session. This does not include instances with the `nlAction=ignore` property that are ignored during netlisting.

Arguments

None

Value Returned

A non-negative integer value representing the number of instances.

Example

```
hnlGetInstanceCount()
```

Cellviews

Cellviews are represented by an object. The `nlGetCurrentCellView` method of the `nlFormatter` class returns such an object. This object is used by formatter methods such as `nlPrintSubcktName` and `nlPrintSubcktParameters`. It is primarily used in the `nlPrintSubcktHeader` and `nlPrintSubcktFooter` methods of the formatter.

Important

Do not redefine any of the methods in this section.

nlGetSimTerminalNets

```
nlGetSimTerminalNets( o_cellView )
     => l_signals
```

Description

Returns the list of mapped names of signals connecting to the terminals of the cellview based on the terminal order specified for the cellview. Several signals may come from inherited connections. Signal buses are handled in scalar form. For example, net10<0:3> is mapped as net10_0, net10_1, net10_2 and net10_3. Use this function while printing the subcircuit definition.

Arguments

o_cellView	The cellview object.

Value Returned

l_signals	Returns the list of mapped signal names connected to the terminals of the subcircuit.

Example

```
nlGetSimTerminalNets( cv )
```

nlGetTerminalNets

```
nlGetTerminalNets( o_cellView )
     => l_signals
```

Description

Returns the schematic names of the signals connected to the terminals. Many of the signals may come from inherited connections.

Arguments

o_cellView	The cellview object.

Value Returned

l_signals	Returns the list of signals connected to the terminals of the subcircuit. Each element is a string.

Example

```
nlGetTerminalNets( cv )
```

nlGetSwitchViewList

```
nlGetSwitchViewList( o_cellView )
     => l_switchViews
```

Description

Returns the switch view list for the cellView.

Arguments

o_cellView	The cellview object.

Value Returned

l_switchViews	Returns the switch view list for the cell view. Each element is a string.

Example

```
nlGetSwitchViewList( cv )
```

Designs

Designs are represented by an object. The `nlGetDesign` method applied on the `netlister` object returns such an object.

Important

Do not redefine any of the methods in this section.

nlGetTopLibName

```
nlGetTopLibName( o_design )
     => t_topLibName
```

Description

Returns the library name of the design.

Arguments

o_design	The design object.

Value Returned

t_topLibName	Returns the library name.

Example

```
nlGetTopLibName( design )
```

nlGetTopCellName

```
nlGetTopCellName( o_design )
     => t_topCellName
```

Description

Returns the cell name of the design.

Arguments

o_design	The design object.

Value Returned

t_topCellName	Returns the cell name.

Example

```
nlGetTopCellName( design )
```

nlGetTopViewName

```
nlGetTopViewName( o_design )
     => t_topViewName
```

Description

Returns the view name of the design.

Arguments

o_design	The design object.

Value Returned

t_topViewName	Returns the view name.

Example

```
nlGetTopViewName( design )
```

nlTranslateFlatIEPathName

```
nlTranslateFlatIEPathName(
     o_formatter
     t_hierDelimiter
     t_iePathName
     )
     => t_iePathName
```

Description

Parses a hierachical IE instance path and returns the path of the IE instance that is to be printed in the digital netlist.

Arguments

o_formatter	The formatter object.
t_hierDelimiter	The delimiter used to parse the hierachical IE instance path.
t_iePathName	The hierachical IE instance path.

Value Returned

t_iePathName	The path of the IE instance to be printed in the digital netlist.

Example

```
defmethod( nlTranslateFlatIEPathName ((formatter yourFormatter) hierDelimiter
iePathName)
iePathComponents = parseString(iePathName hierDelimiter)
yourFunctionToModifyIEPathName(iePathComponents)
)
```

Other Customization procedures

nlSetPcellName

```
nlSetPcellName => t/nil
```

Description

Define this function if the default generic OSS naming convention for pcells needs to be customized.

Arguments

cv	cellView ID of the pcell.
paramNames	pcell parameter Names.
paramValues	pcell parameter Values.

Value Returned

t / nil

Example

Function Name: nlSetPcellName

cv: cellView ID

paramNames: pcell parameter Names

paramValues: pcell parameter Values

Redefine this function to rename your pcell subckt's. In the example below, the subckt's are renamed as `paramName1_paramValue1_paramName2_paramValue2_cellName`

```
procedure( nlSetPcellName( cv paramNames paramValues )
     let( ( name value (subcktName  ""))
     foreach( (name value) paramNames paramValues
          subcktName = strcat( subcktName name "_" value  "_")
     )
    strcat(subcktName cv->cellName )
    )
)
```

auCdl Netlister Functions

ansCdlCompPrim

`ansCdlCompPrim()`

Description

Enables printing of device information for primitives in the auCdl netlist. Specify this function as a netlist procedure in the CDF for the primitive devices for which you want the device information to be printed in the auCdl netlist. For more information about the `ansCdlCompPrim` netlist procedure, see the *Virtuoso Analog Design Environment L User Guide*.

Arguments

None

Value Returned

None

Example

```
ansCdlCompPrim
```

ansCdlHnlPrintInst

```
ansCdlHnlPrintInst()
```

Description

Customizes how device information is written in the auCdl netlist. Specify this function as a netlist procedure in the CDF for the devices for which you want to customize the device information in the auCdl netlist.

Arguments

None

Value Returned

None

Example

```
ansCdlHnlPrintInst
```

ansCdlPrintString

```
ansCdlPrintString(fp inst master parent)
```

Description

Prints comment strings in the device information for instances in the auCdl netlist. To print comment strings in the device information, you must also use the `'string` argument in the `auCdlInstPrintOrder` variable defined in the `.simrc` file. For more information about the `auCdlInstPrintOrder` variable, see the *Virtuoso Analog Design Environment L User Guide*.

Arguments

fp	File handle used to write the string in the netlist using the `artFprintf` SKILL function.
inst	Database ID of the current instance being printed in the netlist.
master	Database ID of the switch master of the current instance being printed in the netlist.
parent	Database ID of the cell for the current instance being printed in the netlist.

Value Returned

None

Example

```
procedure(ansCdlPrintString(fp inst master parent)
artFprintf(fp "*** CurrentInst = %s" inst~>name))
```

ansCdlPrintInheritedParams

```
ansCdlPrintInheritedParams(fp pairList)
```

Description

Customizes how inherited parameters are written in the auCdl netlist.

Arguments

fp	File handle used to write the string in the netlist using the `artFprintf` SKILL function.
pairList	List of property name and property value pairs that should be evaluated.

Value Returned

None

Example

```
procedure(ansCdlPrintInheritedParams(fp pairList)
foreach( param pairList
unless( !cadr(param)
artFprintf( fp "%s=%s " car(param) artMakeString(cadr(param) ) )
)
)

)
```

ansCdlPrintInstParams

```
ansCdlPrintInstParams(fp pairList)
```

Description

Customizes how instance parameters are written in the auCdl netlist.

Arguments

fp	File handle used to write the string in the netlist using the `artFprintf` SKILL function.
pairList	List of property name and property value pairs that should be evaluated.

Value Returned

None

Example

```
procedure( ansCdlPrintInstParams(fp pairList)
        foreach( pair pairList
                artFprintf( fp "%s=%s " car(pair) artMakeString(cadr(pair)) )
            ))
```

ansCdlPrintInstProps

```
ansCdlPrintInstProps(fp pairList)
```

Description

Enables printing of user-defined instance properties and also customizes the format in which the properties are printed in the netlist. To print user-defined properties in the netlist, you must also use the `'instProps` argument in the `auCdlInstPrintOrder` variable defined in the `.simrc` file. For more information about the `auCdlInstPrintOrder` variable, see the *Virtuoso Analog Design Environment L User Guide*.

Arguments

fp	File handle used to write the string in the netlist using the `artFprintf` SKILL function.
pairList	List of property name and property value pairs that should be evaluated.

Value Returned

None

Example

```
procedure( ansCdlPrintInstProps(fp pairList)
        foreach( pair pairList
                artFprintf( fp "%s=%s " car(pair) artMakeString(cadr(pair)) )
            ))
```

ansCdlPrintInstName

```
ansCdlPrintInstName(fp prefix name mappedName isPrimitive inst master)
```

Description

Customizes how instance names are written in the auCdl netlist.

Arguments

fp	File handle used to write the string in the netlist using the `artFprintf` SKILL function.
prefix	Name prefix defined in the device CDF.
name	Instance name specified for the instance.
mappedName	Mapped name which auCdl would otherwise use to print instance name.
isPrimitive	Whether this is an instance of a leaf-level device (primitive).
inst	Database ID of the current instance being printed in the netlist.
master	Database ID of the switch master of the current instance being printed in the netlist.

Value Returned

None

Example

```
procedure( ansCdlPrintInstName(fp prefix name mappedName isPrimitive inst master)
        artFprintf(fp "%s " mappedName ))
```

ansCdlPrintModelName

```
ansCdlPrintModelName(fp isAPrimitive definedPropVal modelPropInstVal
     componentPropInstVal cdfModelName cdfComponentName)
```

Description

Customizes the order in which auCdl looks for model names for primitives and the format in which the model information is written in the netlist.

Arguments

fp	File handle used to write the string in the netlist using the `artFprintf` SKILL function.
isPrimitive	Whether this is an instance of a leaf-level device(primitive).
definedPropVal	Value of the instance property defined using `auCdlHnlInstModelPropName` parameter in the `.simrc` file. For more information about the `auCdlHnlInstModelPropName` parameter, see the *Virtuoso Analog Design Environment L User Guide*.
modelPropInstVal	Value of the `'model` property on the instance.
componentPropInstVal	Value of the `'componentName` property on the instance.
cdfModelName	Value of the `'modelName` parameter in the device CDF.
cdfComponentName	Value of the `'componentName` parameter in the device CDF.

Value Returned

None

Example

```
procedure( ansCdlPrintModelName(fp isAPrimitive definedPropVal
                                modelPropInstVal
                                componentPropInstVal
                                cdfModelName
                                cdfComponentName
                           )
```

```
when( isAPrimitive
    let( (model)
        model = nil

        if( definedPropVal
            model  = definedPropVal
          )
        if( !model && modelPropInstVal
            model =  definedPropVal
          )
        if( !model && cdfModelName
            model = cdfModelName
          )
        if( !model && componentPropInstVal
            model = componentPropInstVal
          )
        if( !model && cdfComponentName
            model = cdfComponentName
          )
            artFprintf(fp "model=%s " artMakeString( model ) )
        )
    ))
```

ansCdlPrintModuleName

```
ansCdlPrintModuleName(fp isAPrimitive inst master parent mappedModuleName)
```

Description

Customizes how module names are written in the auCdl netlist for subcircuits.

Arguments

fp	File handle used to write the string in the netlist using the `artFprintf` SKILL function.
isAPrimitive	Whether this is an instance of a leaf-level device(primitive).
inst	Database ID of the current instance being printed in the netlist.
master	Database ID of the master device for the current instance being printed in the netlist.
parent	Database ID of the cell for the current instance being printed in the netlist.
mappedModuleName	Module name which auCdl would write by default for subcircuits.

Value Returned

None

Example

```
procedure( ansCdlPrintModuleName(fp isAPrimitive inst master parent
mappedModuleName)
        unless( isAPrimitive
        artFprintf( fp " / %s " mappedModuleName )
        ))
```

ansCdlPrintConnections

```
ansCdlPrintConnections(fp connections)
```

Description

Customizes how the nets connected to a device are written in the auCdl netlist.

Arguments

fp	File handle used to write the string in the netlist using the `artFprintf` SKILL function.
connections	List of pairs of instance terminals and nets connected to the terminals.

Value Returned

None

Example

```
procedure( ansCdlPrintConnections(fp connections )
        artFprintf( fp "$PINS " )
        foreach( cn connections
            artFprintf( fp "%s=%s " car(cn) cadr(cn) )  ))
```

ansCdlGetSegmentConnections

```
ansCdlGetSegmentConnections( inst connectionPairs iterSeg iterMult numSegments
     multiplicityFactor segmentConnType netCount )
```

Description

This function is used to customize the auCdl netlist when the `ansCdlHnlPrintInst` function is specified as a netlist procedure in the CDF for the device. It controls how connectivity information is written in the netlist for instances for which a multiplicity factor is specified using the `m` or `M` property. For example, if an instance with a connection list like `((termA netA) (termB netB))` has to be converted into two segments connected in series, the modified connection list for the first segment will be `((termA netA) (termB tempnet_0))` and `((termA tempnet_0) (termB netB))` for the second segment. Define this function as a procedure in the `.simrc` file.

Arguments

inst	Database ID of original instance.
connectionPairs	List of pairs of instance terminals and nets connected to the terminals of the instance.
multiplicityFactor	Number of replications of the instance.
numSegments	Number of segments in each replication of the instance.
iterMult	If the current instance needs to be converted into 5 instances connected in parallel, then this function is called for which iteration is from 1 - 5.
iterSeg	If current replication needs to be converted into 3 segments connected in series or parallel, then this function is called for which iteration is from 1- 3.
segmentConnType	Connection type of the segments. Valid values: `"series"` or `"parallel"` or any other user defined value.
netCount	A pretty ordinary number which keeps on incrementing with calls to this function. This will help you create temporary nets to connect segments in series.

Value Returned

None

Example

```
procedure( ansCdlGetSegmentConnections( inst connectionPairs iterSeg iterMult
numSegments multiplicityFactor segmentConnType netCount  )
    let( ( firstTerm secondTerm pairList )
    if( "series" == segmentConnType
        then
        cond(
        ( (1 == iterSeg )
        secondTerm = cadr( connectionPairs )
        secondTerm = list( car(secondTerm) sprintf( nil "CdnsNet_%d"
            netCount) )
        pairList = list( car(connectionPairs) secondTerm )
        pairList = append( pairList  cddr(connectionPairs) )
        )
        ( ( iterSeg == numSegments )
        firstTerm = car(connectionPairs)
        firstTerm = list( car(firstTerm) sprintf(nil "%s_%d" "CdnsNet"
            netCount-1 ) )
        pairList = append( list(firstTerm) cdr(connectionPairs))
        )
        (t
        firstTerm = car(connectionPairs)
        secondTerm = cadr(connectionPairs)
        firstTerm = list( car(firstTerm) sprintf(nil "CdnsNet_%d"
            netCount-1) )
        secondTerm = list( car(secondTerm) sprintf(nil "CdnsNet_%d"
            netCount ) )
        pairList = append1( list(firstTerm) secondTerm )
        pairList = append( pairList cddr(connectionPairs))
        )
        )
        else
        pairList = connectionPairs
    )
    pairList
    )
```

```
)
```

Related Functions

- ansCdlGetMultiplicity on page 168
- ansCdlGetSegmentInfo on page 163

ansCdlGetSegmentInfo

```
ansCdlGetSegmentInfo( inst master parent )
```

Description

This function is used to customize the auCdl netlist when the `ansCdlHnlPrintInst` function is specified as a netlist procedure in the CDF for the device. It returns the number of segments for an instance and the connection type (`series`, `parallel`, or user defined connection type name) between the segments. Define this function as a procedure in the `.simrc` file.

Arguments

inst	Database ID of the current instance being printed in the netlist.
master	Database ID of the switch master of the current instance being printed in the netlist.
parent	Database ID of the cell for the current instance being printed in the netlist.

Value Returned

Number of segments for the instance and the connection type (`series`, `parallel`, or user defined) between segments.

Example

```
procedure( ansCdlGetSegmentInfo( inst master parent)
        prog( (val type )
           val = ansCdlGetSimPropValue( 'segments )
           when( val
                 when( stringp(val) val = atoi( val) )
                 when( numberp(val) val = fix(val) )
            )
           unless( val return() )
           type = ansCdlGetSimPropValue( 'connection )
           when( type && symbolp( type)
               type = symbolToString( type ) )
            return( list( val type))
```

```
        )
    )
```

Related Function

- ansCdlGetMultiplicity on page 168
- ansCdlGetSegmentConnections on page 160

ansCdlGetSegmentInstParams

```
ansCdlGetSegmentInstParams( inst propsList iterSeg iterMult numSegments
multiplicityFactor segmentConnType )
```

Description

This function is used to customize the auCdl netlist when the `ansCdlHnlPrintInst` function is specified as a netlist procedure in the CDF for the device. It controls how values of parameters are printed on segments of an instance. auCdl calls this function with a list of property name and property value pairs for all those properties that are specified in device CDF simulation information instance parameters. Define this function as a procedure in the `.simrc` file.

Arguments

inst	Database ID of original instance.
propsList	List of property name and property value pairs that should be evaluated.
iterSeg	If current replication needs to be converted into 3 segments connected in series or parallel, then this function is called for which iteration is from 1- 3.
iterMult	If the current instance needs to be converted into 5 instances connected in parallel, then this function is called for which iteration is from 1 - 5.
numSegments	Number of segments in each replication of the instance.
multiplicityFactor	Number of replications of the instance.
segmentConnType	Connection type of the segments. Valid values: `"series"` or `"parallel"` or any other user defined value.

Value Returned

A modified list of property name and property value pairs for each segment. By default, auCdl divides the length of the instance by the number of segments in case of series connection, and divides the width of the instance by the number of segments in case of parallel connection.

Example

```
procedure( ansCdlGetSegmentInstParams( inst propsList iterSeg iterMult
    numSegments multiplicityFactor segmentConnType )
    let((pairList value)
    foreach( pair propsList
        case( car(pair)
        ("l"
        value = cadr(pair)
        when( "series" == segmentConnType
            when( stringp(value)
            value = atoi(value) )
            value = value/numSegments )
        pairList = cons( list( "l" value) pairList )
        )
        ("w"
        value = cadr(pair )
        when( "parallel" == segmentConnType
            when( stringp(value)
            value = atoi(value))
            value = value/numSegments )
        pairList = cons( list( "w" value ) pairList )
        )
        (t
        pairList = cons( pair pairList )
        )
        )
        )
    pairList = reverse( pairList )
    )
    )
```

Related Function

- ansCdlGetMultiplicity on page 168
- ansCdlGetSegmentInfo on page 163

ansCdlGetSimPropValue

```
ansCdlGetSimPropValue ( t_propName )
     => t_propVal/nil
```

Description

Returns the value of a specified property on the current instance being netlisted when the `ansCdlHnlPrintInst` function is specified as a netlist procedure in the CDF for the device.

Arguments

t_propName	Name of the property on the current instance being netlisted for which the value is to be returned.

Value Returned

t_propVal	Returns the value corresponding to a property.
nil	Returns *nil* otherwise.

Example

```
ansCdlSimGetPropValue( 'vendorName )
```

ansCdlGetMultiplicity

```
ansCdlGetMultiplicity( inst master parent )
     => x_multiplicityFactor
```

Description

Controls how multiplicity (converting an instance into multiple instances connected in parallel) is handled for an instance in the design when the `ansCdlHnlPrintInst` function is specified as a netlist procedure in the CDF for the master of the instance.

You can specify that an instance should be treated as *n* instances connected in parallel by specifying *n* as the value of the `m` or `M` property on the instance. For example, you can specify that an instance be treated as five instances connected in parallel by specifying `5` as the value of the `m` or `M` property on the instance. By default the `ansCdlHnlPrintInst` netlist procedure does not give special treatment to the `m` or `M` property on an instance.

If you want the `ansCdlHnlPrintInst` netlist procedure to support multiplicity on instances using the `m` or `M` property, do one of the following:

- Define the `ansCdlGetMultiplicity` procedure in your `.simrc` file.
- Define `ansCdlGetMultiplicity_<`*LIBNAME*`>` in `libInit.il` file of the device library when the `libSpecificDevicePrint` flag is set in device `instParameters`.

Arguments

inst	Database ID of the current instance being printed in the netlist.
master	Database ID of the switch master of the current instance being printed in the netlist.
parent	Database ID of the cell for the current instance being printed in the netlist.

Value Returned

x_multiplicityFactor	Multiplicity factor.

Example

```
procedure( ansCdlGetMultiplicity( inst master parent )
     ansCdlGetSimPropValue( 'm )
     )
```

Related Function

- ansCdlGetSegmentInfo on page 163
- ansCdlGetSegmentConnections on page 160
- ansCdlGetSegmentInstParams on page 165

auCdl

`auCdl()`

Description

This function sets defaults of all the variables defined in `si.env` like `preserveRES`, `shortRES` as well as other global variables like `cdlSimViewList`, `cdlPrintComments` etc. The function also sets the list of functions and variables that must be unbound when environments(simulators) are changed.

Arguments

None

Values Returned

`t` or `nil` is returned

Other Backend Netlister Functions

auLvs

`auLvs()`

Description

This is the primary function for LVS. It sets up all the actions needed to netlist the layout and schematics design as well as invoke LVS (for comparison) itself.

Argument

None

Values Returned

Returns `t` or `nil` based on the status of the netlisting and the status returned by the LVS routine which carries out the comparison between the netlist from the schematic view and the layout view. For any error, `nil` is returned and for a successful netlisting and comparison `t` is returned with the relevant messages printed in the UI and the `si.log` file.

Example

```
auLvs()
```

auProbeAddDevsForNet

```
auProbeAddDevsForNet()
```

Description

This procedure enables you to select nets using the cursor or by typing the names in the CIW, in order to add probes for all devices connected to the selected net. Thus, the function displays the prompt, `Point to net or enter net name in CIW`. On pointing to the net or typing the net name in CIW, probes would be added on all the devices connected to the selected net.

Argument

None

Values Returned

`t` or `nil` is returned

LVS

`LVS()`

Description

This is the primary function for LVS. It sets up all the actions needed to netlist the layout and schematics design as well as invoke LVS itself.

Arguments

None

Value Returned

`t` or `nil` is returned

Example

LVS()

HSPICE Functions

hnlHspicePrintInstPropVal

```
hnlHspicePrintInstPropVal( t_propName ) => t_propVal | nil
```

Description

This procedure returns the value of the property specified by propName if it exists on the current instance. If the value of this property has the syntax specifying an inherited value it simply returns the name of the property whose value is being inherited without the surrounding syntax.

Arguments

t_propName	Specifies the property name

Value Returned

t_propVal	Returns the value corresponding to a property. The return value for this procedure is always a string.
nil	Returns *nil* otherwise.

hnlHspiceInstPropVal

```
hnlHspiceInstPropVal( l_paramList ) => t | nil
```

Description

This procedure is prints a list of property values to the netlist.

Arguments

l_paramList	List of strings containing the property names.

Value Returned

t	Returns *t* if the property values are successfully printed to the netlist.
nil	Returns *nil* otherwise.

hnlHspicePrintInstPropEqVal

```
hnlHspicePrintInstPropEqVal( l_paramList ) => t | nil
```

Description

This procedure prints a list of property values to the netlist. It is similar to the procedure `hnlHspicePrintInstPropVal` except that the property name and the symbol, `=` is included before the value.

Arguments

`l_paramList`	List of strings containing the property names.

Value Returned

`t`	It returns *`t`* if the property values are successfully printed to the netlist.
`nil`	Returns *`nil`* otherwise.

hnlHspicePrintMOSfetModel

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

Description

This function prints out the line for a MOSfet model.

Arguments

None

Value Returned

t	It returns *t* if the mosfet model is successfully printed to the netlist file.
nil	Returns *nil* otherwise.

hnlHspicePrintNMOSfetElement

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

Description

This function prints out the line for a NMOSfet model.

Arguments

None

Value Returned

t	It returns *t* if the nmosfet model is successfully printed to the netlist file.
nil	Returns *nil* otherwise.

Name Mapping Variables

Spectre

Described below are the name mapping variables for Spectre Direct netlister:

Variable	Description
hnlSpectreMapInstInName	List of characters that are invalid internal to an inst name.
hnlSpectreMapNetInName	List of characters that are invalid internal to a net name.
hnlSpectreMapInstFirstChar	List of characters that are invalid for the first character of a inst name.
hnlSpectreMapNetFirstChar	List of characters that are invalid for the first character of a net name.

Default Behavior

- Replace the invalid characters by their escaped equivalent.
- If the invalid character is a number, replace it by an underscore followed by the number.

Customization

- You can directly set any/all of the above mentioned variables to customize the character substitution and therefore the mapping.
- Each of these variables is checked separately to determine whether it is to be set or not.
- In case a variable is set, it is used to determine character substitution and hence mapping.

Flow

- The variables are used if they are set explicitly.
- Otherwise, the first time you netlist, all the variables are populated with their default values. The second time onwards the variable values will be used. If you need to modify just a part of the character substitution list, you can do it by modifying the variable in the CIW window. Otherwise the default values are used throughout.

HspiceD

Described below are the name mapping variables for HspiceD netlister:

Variable	Description
hnlHspiceDMapInstInName	List of characters that are invalid internal to an inst name.
hnlHspiceDMapNetInName	List of characters that are invalid internal to a net name.

Default Behavior

These variables do not have default values.

Customization

- You can directly set any/all of the above mentioned variables to customize the character substitution and therefore the mapping.
- Each of these variables is checked separately to determine whether it is to be set or not.
- In case a variable is set, it is used to determine character substitution and hence mapping.

Flow

- The variables are used if they are set explicitly.
- If these variables are not set explicitly, HspiceD netlister uses the following values:
  - For mapping nets, it uses:

    (">" ">") ("<" "<") "." ("#" "__") "$" "%" "^" "&" "*" "(" ")" "\\" ("|" "__") "+" "-" "=" "{" "}" ("[" "__") ("]" "__") "\"" ("'" "__") (":" "__") ";" "~" ("`" "__") "," "?" "/" "@")
  - For mapping instances, it uses:

    (">" ">") ("<" "<") "." "!" ("#" "__") "$" "%" "^" "&" "*" "(" ")" "\\" ("|" "__") "+" "-" "=" "{" "}" ("[" "__") ("]" "__") "\"" "'" (":" "__") ";" "~" "`" "," "?" "/" "@")

5

Netlisting Option Functions for Socket Interfaces

This chapter describes the functions that let you work with netlist options for socket interfaces.

asiDisplayNetlistOption

```
asiDisplayNetlistOption( o_tool )
     => t | nil
```

Description

Displays the current set of netlist options and values. Use this function only to determine which netlist options you can modify. Do not use this function as part of another procedure.

Arguments

o_tool	Simulation tool object.

Value Returned

`t`	Displays the set of netlist option names and values and returns `t`.
`nil`	Prints an error message and returns `nil` if there is an error.

Example

```
asiDisplayNetlistOption( asiGetTool( 'spectreS ) )
```

Displays the *spectreS* netlist options.

asiGetNetlistOption

```
asiGetNetlistOption(
     {o_session | o_tool}
     s_name
     )
     => g_value | nil
```

Description

Gets the value of the specified netlist option.

Arguments

o_session	Simulation session object.
o_tool	Simulation tool object.
s_name	Netlist option for which you want the value.

Value Returned

g_value	Returns the value of the netlist option.
nil	Returns nil if there is an error.

Example

```
asiGetNetlistOption( session 'modelNamePrefix )
```

Gets the value of the modelNamePrefix netlist option.

Related Function

To display the current set of netlist options, see the asiDisplayNetlistOption function on page 5-184.

asiInit<yourSimulator>NetlistOption

```
asiInit<yourSimulator>NetlistOption( o_tool )
     => t
```

Description

Calls the procedures that modify your simulator's netlist options.

Note: You must write `asiInit<yourSimulator>NetlistOption`, where `<yourSimulator>` is the name of your simulator. Do not include the angle brackets (<>).

Arguments

o_tool	Simulation tool object.

Value Returned

`t`	Returns `t` when your procedures for modifying netlist options are called.

Note: You must write `asiInit<yourSimulator>NetlistOption` to return `t`.

Example

```
procedure( asiInitXYZNetlistOption( tool )
    <insert your code>
    t
)
```

Creates the procedure that calls the procedures to modify the netlist options for the XYZ simulator.

asiSetNetlistOption

```
asiSetNetlistOption(
     {o_session | o_tool}
     s_name
     g_value
     )
     => g_value | nil
```

Description

Sets a netlisting option.

Arguments

o_session	Simulation session object.
o_tool	Simulation tool object.
s_name	Name of the netlist option you want to set.
g_value	Value for the netlist option.

Value Returned

g_value	Returns the value of the netlist option.
nil	Returns an error message and nil if unsuccessful.

Example

```
asiSetNetlistOption( session 'maxNameLength 16 )
```

Changes the maximum length of the net name to 16.

Related Function

To display the names of the netlist options that you can set, see the asiDisplayNetlistOption function on page 5-184.

6

OASIS Functions

This chapter describes the functions that let you print and manage OASIS files.

asiGetAnalogSimulator

```
asiGetAnalogSimulator( { o_session | o_tool } )
     => s_simulatorName
```

Description

Gets the value of the analog simulator for a tool or session object.

Arguments

o_session	Simulation session object.
o_tool	Simulation tool object.

Value Returned

s_simulatorName	Name of the analog simulator.

Example

```
asiGetAnalogSimulator(asiGetTool('spectreS))
```

Displays the analog simulator for the tool spectreS as `spectreS`.

```
asiGetAnalogSimulator(asiGetTool('spectreVerilog))
```

Displays the analog simulator for the tool spectreVerilog as `spectre`.

asiGetDigitalSimulator

```
asiGetDigitalSimulator( { o_session | o_tool } )
     => s_simulatorName / nil
```

Description

Gets the value of the digital simulator for a tool or session object.

Arguments

o_session	Simulation session object.
o_tool	Simulation tool object.

Value Returned

s_simulatorName	Name of the digital simulator.
nil	Returns nil if there is no digital simulator.

Example

```
asiGetDigitalSimulator(asiGetTool('spectreVerilog))
```

Displays the digital simulator for the tool spectreVerilog as verilog.

```
asiGetDigitalSimulator(asiGetTool('spectreS))
```

Displays the digital simulator for the tool spectreS as nil as there is no digital simulator for spectreS.

asiAnalogAutoloadProc

```
asiAnalogAutoloadProc() => t
```

Description

Called by OASIS for the purpose of autoloading the context. This is done so that the classes are defined before the tool is created and initialization is started.

Argument

None

Value Returned

t	The context is autoloaded successfully.

Example

```
asiAnalogAutoloadProc
```

ansAnalogRegCDFsimInfo

```
ansAnalogRegCDFsimInfo() => t
```

Description

This is a utility function used to create the CDF for the <yourSimulator> simulator.The ansAnalogRegCDFsimInfo functions are called by the CDF editor. These functions are used to provide data type information for all the simInfo attributes.

Argument

None

Return Value

t	Function defined successfully.

Example

The following illustrates the declaration of ansAnalogRegCDFsimInfo:

```
procedure( ansAnalogRegCDFsimInfo()
    '( (nil name netlistProcedure type symbol)
        (nil name otherParameters type list)
        (nil name instParameters type list)
        (nil name componentName type symbol)
        (nil name namePrefix type string)
        (nil name termOrder type list)
        (nil name termMapping type list)
        (nil name propMapping type list)
    )
)
```

The following is a sample <yourSimulator>.ini file for <yourSimulator> simulator:

```
asiRegisterTool( '<yourSimulator> ?class 'asiAnalog ?initFunc
'asiInitialize)
procedure( ansRegCDFsimInfo_<yourSimulator>()
    ansAnalogRegCDFsimInfo()
)
```

Note: To assure consistency in netlisting across simulators, the parameters returned by `ansAnalogRegCDFsimInfo` must not be redefined. If the formatter of your simulator has the `useInstNamePrefix netlist` option set to `nil`, the namePrefix parameter must be removed.

asiCheckAcEnabledWhenNoiseEnabled

```
asiCheckAcEnabledWhenNoiseEnabled( o_session r_form ) => t | nil
```

Description

This method verifies that an AC analysis is enabled when a noise analysis is selected. If this is not the case, it displays an error message in the analysis form's error dialog box and sets the error status of the form. It is called during the form apply callback.

Arguments

o_session	The simulation session object.
r_form	Form created by a call to hiCreateAppForm or hiCreateForm.

Values Returned

t	Returns true when check is successful
nil	Returns an error message and nil if unsuccessful

asiCheckAnalysis

```
asiCheckAnalysis( o_analog r_form ) => t | nil
```

Description

Checking function for the analysis class. e.g. for Spectre simulator object it checks each field value in the environment, highlights any errors, returns t/nil. Can be used for different analog simulator analysis objects.

> **Note:** The only difference between this and the check method in the analog class, is that this one does not give an error if 'step' is not set.

Arguments

o_analog	derived object of the analog simulator class
r_form	form created by a call to hiCreateAppForm or hiCreateForm.

Values Returned

t	Returns true when check is successful
nil	Otherwise

asiCheckBlank

```
asiCheckBlank( o_obj r_form s_fieldName ) => t | nil
```

Description

Verifies that the *s_fieldName* entry is non-blank.

Arguments

o_obj	Can be one of the following objects: `o_analysis`, `o_anaOption`, `o_envOption`, `o_simOption`, `o_keepOption`.
r_form	Form created by a call to `hiCreateAppForm` or `hiCreateForm`.
s_fieldName	Handle to this field within a form. It may be referenced as `formHandle->hiFieldSym`

Values Returned

`t`	Check is successful
`nil`	Otherwise

asiFormatMTSModelAndSimOptions

```
asiFormatMTSModelAndSimOptions( o_session fp ) => t | nil
```

Description

This method should be overridden to alter the format of the include file for an MTS block. The included file contains MTS options: Simulator Options and Model Files. The default implementation is in spectre format and is formatted as: simulatorOptions options tnom=27 scalem=1 scale=2 include Models/myModels.scs.

Arguments

o_session	The OASIS session object.
fp	Include file pointer. This file pointer is handled by the internal OASIS code. It should only be used to print the MTS block options.

Values Returned

t	The call is successful
nil	The call has failed

Example

For the default analog session the current implementation is:

```
(defmethod asiFormatMTSModelAndSimOptions (( session asiAnalog_session )
fp options )
    let( ((firstOption t) simOption modelFile sectionName text )
        foreach( simOption get( options 'simOptions )
        (when firstOption
            ;; Change the "simulator Options" text below, if target
simulator uses a
            ;; different syntax
    (artFprintf fp "simulatorOptions options ")
    firstOption=nil
    )
    (artFprintf fp "%s=%s " car(simOption) cadr(simOption))
    )
```

```
        (artFprintf fp "\n")
        foreach( modelFile get( options 'modelFiles )
            sectionName = cadr( modelFile )
            if( and( sectionName !artBlankString( sectionName ))
                text = strcat("\"" car(modelFile) "\"" " section="
sectionName)
                text = strcat("\"" car(modelFile) "\"")
            )
        (artFprintf fp asiMTSIncludeFormat( session ) text )
        )
    )
    t
)
```

asiGetAnalysisField

```
asiGetAnalysisField( o_analysis s_fieldName )  => o_fieldEnvVar | nil
```

Description

Returns the specified analysis field object.

Arguments

o_analysis	The analysis object for which you want the field object.
s_fieldName	Handle to this field within a form. It may be referenced as `formHandle->hiFieldSym`

Values Returned

o_fieldEnvVar	Object of field environment variable
nil	Returns `nil` if no analysis field is present.

Example

```
analysis = asiGetAnalysis( session 'tran )
asiGetAnalysisField( analysis 'stop )
```

Gets the tran analysis, stop field object.

asiGetHighPerformanceOptionVal

```
asiGetHighPerformanceOptionVal( o_session s_varName )  => t_varNameVal | nil
```

Description

Returns the field value of the High Performance form with the passed session object.

Note: This method returns the field value of the High Performance form for spectre class (*spectre_session*) only. For other classes, this method returns nil.

Arguments

o_session	The simulation session object.
s_varName	The field name of the High Performance form.

Values Returned

t_varNameVal	The value of the specified field.
nil	Returns `nil` when function call unsuccessful.

Example

```
session = asiGetCurrentSession()
asiGetHighPerformanceOptionVal( session 'uniMode )
```

Returns the value of uniMode field for spectre class.

asiSetHighPerformanceOptionVal

```
asiSetHighPerformanceOptionVal( s_sessionName s_varName s_varValue )
     => t_Value | nil
```

Description

Sets the value for the specified High Performance option variable.

Arguments

o_sessionName	The session for the simulator.
s_varName	Name of the variable in the High Performance Option form that you want to set.
s_varValue	The value of the variable field.

Values Returned

t_Value	Returns the value of the specified field.
nil	Returns `nil` if unsuccessful.

Example

```
Session = asiGetCurrentSession()
asiSetHighPerformanceOptionVal(Session 'uniMode "APS")
=> "APS"
```

Sets the value of uniMode field to APS for spectre class.

asiDisplayHighPerformanceOption

```
asiDisplayHighPerformanceOption( o_toolName )
     => t | nil
```

Description

Displays a list of variable and values of the High Performance Simulation Option form.

Arguments

o_toolName	The tool object according to the simulator.

Values Returned

t	Displays the list of options and values of the High Performance Simulation Option form and returns `t`.
nil	Returns `nil` if unsuccessful.

Example 1

```
asiDisplayHighPerformanceOption(asiGetTool('spectre))
    uniMode:    "Spectre"
    turboSwitch:    nil
    envSwitch:    nil
    uniSeparate:
    errorLevel:    "Do not override"
    mtOption:    "Auto"
    numThreads:    ""
    apsplus:    nil
    cktpreset:    "None"
    proc_affinity:    ""
    pseparate:
    srSwitch:    nil
    psrOption:    "Default"
    psrFmax:    ""
    preserveOption:    "None"
    preserveInst:    nil
    preserveSelect:
    preserveClear:
    rfseparate:
    rfmtOption:    "Disable"
    numRFThreads:    ""
```

Example 2

```
asiDisplayHighPerformanceOption(asiGetTool('ams))
=> ERROR (ADE-5032): asiEnvDisplayVar: No variables defined in partition
'turboOpts' of tool 'ams'.
nil
```

asiGetDesignCellName

```
asiGetDesignCellName( o_session ) =>t_cellName | nil
```

Description

Returns the cell name of the design associated with the passed session object.

Arguments

o_session	The simulation session object.

Values Returned

t_cellName	Name of the cell associated with the session obj.
nil	The function failed to give cell name.

Example

```
session = asiGetCurrentSession()
asiGetDesignCellName(session)
```

asiGetDesignLibName

```
asiGetDesignLibName( o_session )=> t_libName | nil
```

Description

Returns the library name of the design associated with the passed object.

Arguments

o_session	The simulation session object.

Values Returned

t_libName	Name of the library containing the cellview
nil	Returns nil when func call fails.

Example

```
session = asiGetCurrentSession()
asiGetDesignLibName(session)
```

asiGetDesignViewName

```
asiGetDesignViewName( o_session ) => t_viewName | nil
```

Description

Returns the view name of the design associated with the passed object.

Arguments

o_session	The simulation session object.

Values Returned

t_viewName	Name of the view related with the passed session object
nil	Returns `nil` when func call fails.

Example

```
session = asiGetCurrentSession()
asiGetDesignViewName(session)
```

asiGetDrlData

```
asiGetDrlData( t_anaType l_specifier t_dataDir ) => g_familyOrWaveform | nil
```

Description

Returns the results data for the specifier specified by the data directory.

Arguments

t_anaType	Name of the analysis type
l_specifier	Specifier name list
t_dataDir	Data directory name

Values Returned

g_family	Family data
Waveform	Waveform data
nil	Otherwise

Note: This function is required for 3rd party simulator integrations. For related information, you can also refer to the *asiDefineDataAccess* function.

asiGetId

```
asiGetId( o_session ) => x_id
```

Description

Returns the name of the session ID associated with the given OASIS session.

Arguments

o_session	The simulation session object.

Values Returned

x_id	OASIS session ID.

Example

```
asiGetId( asiGetCurrentSession() ) => 1
```

Gets the ID of the current OASIS session.

asiGetResultsPsfDir

```
asiGetResultsPsfDir(
     o_session
     )
     => t_PsfDir | nil
```

Description

Returns the name of the PSF directory for the current or last-run simulation.

Arguments

o_session	The oasis session.

Value Returned

t_PsfDir	Returns the path of the PSF directory for the last simulation results.
nil	Returns nil if the directory is not present.

Examples

The following example returns the PSF directory for the current simulation session.

```
asiGetResultsNetlistDir(asiGetCurrentSession())
=> "/servers/scratch02/aakhil/testcases/newtest/simulation/ampTest/spectre/
schematic/distributed/job026/psf"
```

asiGetResultsNetlistDir

```
asiGetResultsNetlistDir(
     o_session
     )
     => t_netlistDir | nil
```

Description

Returns the name of the netlist directory for the current or last-run simulation.

Arguments

o_session	The oasis session.

Value Returned

t_netlistDir	Returns the path of the netlist directory for the last simulation results.
nil	Returns nil if the directory is not present.

Examples

The following example returns the netlist directory for the current simulation session.

```
asiGetResultsNetlistDir(asiGetCurrentSession())
=> "/servers/scratch02/aakhil/testcases/newtest/simulation/ampTest/spectre/
schematic/distributed/job026/netlist"
```

asiGetSimulatorList

```
asiGetSimulatorList( @optional s_subclass )
     => l_simulatorNameList | nil
```

Description

The function returns a list of all simulation interfaces within the specified simulator subclass.

Arguments

s_subclass	Name of the sub-class containing the OASIS simulation interfaces.

Return Value

`l_simulatorNameList`	List containing simulation interfaces associated with the simulator subclass.
`nil`	Returns `nil` if there are no entries.

Example

```
asiGetSimulatorList()
("ats" "cdsSpice" "cdsSpiceVerilog" "hspiceS" "hspiceSVerilog"
"spectre" "spectreS" "spectreSVerilog" "spectreVerilog"
)
```

For specific subclasses like asiAnalog & asiSocket this function returns the corresponding simulatorNamelist.

```
asiGetSimulatorList('asiAnalog)
("ats" "cdsSpice" "cdsSpiceVerilog" "hspiceS" "hspiceSVerilog"
    "spectre" "spectreS" "spectreSVerilog" "spectreVerilog"
)
asiGetSimulatorList('asiSocket)
("cdsSpice" "cdsSpiceVerilog" "hspiceS" "hspiceSVerilog" "spectreS"
    "spectreSVerilog"
)
```

asiGetSimCommandLineOrder

```
asiGetSimCommandLineOrder( o_session )
     => s_optionList | nil
```

Description

Returns the order of options used in the simulator run command. By default, it returns the options in the following order:

```
simulatorName inputFile simOptions scriptOptions
```

To get a different order of the options in the list returned, overload the function to specify the desired order. For example,

```
defmethod( asiGetSimCommandLineOrder( o_session )
'(simulatorName simOptions scriptOptions inputFile)
)
```

Argument

o_session	Simulation session object.

Value Returned

s_optionList	List of the options in the default or specified order.
nil	Returns nil otherwise.

Example

```
asiGetSimCommandLineOrder( spectre_seesion )
```

Displays the options used to run the spectre simulator.

asiGetStimulusGlobals

```
asiGetStimulusGlobals(o_session) => l_globals | nil
```

Description

This method retrieves the list of global stimuli from the session.

Arguments

o_session	Specifies the session object.

Values Returned

l_globals	List of global stimuli.
nil	Returns nil if there is no global stimuli associated with the current session.

asiGetStimulusInputs

`asiGetStimulusInputs(`*`o_session`*`) => `*`l_inputs`*` | nil`

Description

This method retrieves the list of input stimuli from the session.

Arguments

`o_session`	specifies the session object.

Value Returned

`l_inputs`	List of input stimuli.
`nil`	Returns `nil` if there is no input stimuli associated with the current session.

asiIsConfigDesign

```
asiIsConfigDesign( o_session ) => t | nil
```

Description

The function returns true if the design associated with the session is a Cadence 5.x configuration.

Arguments

o_session	The simulation session object.

Values Returned

`t`	When the design belongs to `5.x` config.
`nil`	Returns `nil` otherwise.

Example

```
asiIsConfigDesign(session)
```

asiMTSIncludeFileExtension

```
asiMTSIncludeFileExtension( o_session ) => t_extension
```

Description

Returns file extension of MTS block specific include file so that the `spectre (.scs )` and `spice (.c )` syntax can be taken care of. The default extension is `.scs.`

Arguments

o_session	The OASIS session object.

Values Returned

t_extension	The file extension

asiMTSIncludeFormat

```
asiMTSIncludeFormat( o_session ) => t_syntax | nil
```

Description

This method should be overridden to alter the format of the include statement that goes into the netlist. The default include statement is in the spectre syntax and is formatted as: include mts-opamplib-amplifier.scs.

Arguments

o_session	The OASIS session object.

Values Returned

t_syntax	Format of the string
nil	The call has failed

Example

For the default analog session the current implementation is:

```
(defmethod asiMTSIncludeFormat (( session asiAnalog_session )) "include %s\n" )
```

asiSetValid

```
asiSetValid( o_analysis ) => t | nil
```

Description

The functions sets valid analysis in the current simulation environment. The call to `asiSetValid` should be used with `asiCheck`/`asiCheckAnalysis` method for the analysis.

Arguments

o_analysis	The analysis object used
t/nil	The value you want to choose either t or nil.

Values Returned

`t`	When the function sets value as true
`nil`	Otherwise

asiCheckBlankNumericLeq

```
asiCheckBlankNumericLeq( o_obj r_form s_fieldName g_value ) => t | nil
```

Description

Verifies that the s_fieldName entry is a numeric value less than or equal to g_value.

Arguments

o_obj	Can be one of the following objects: o_analysis, o_anaOption, o_envOption, o_simOption, o_keepOption.
r_form	Form created by a call to hiCreateAppForm or hiCreateForm.
s_field	NameHandle to this field within a form. It may be referenced as formHandle->hiFieldSym
g_value	The value you want to choose.

Values Returned

t	When the check is successful.
nil	Otherwise

asiCheckBlankNumericGeq

```
asiCheckBlankNumericGeq( o_obj r_form s_fieldName g_value ) => t | nil
```

Description

Verifies that the *s_fieldName* entry is a numeric value greater than or equal to *g_value*.

Arguments

o_obj	Can be one of the following objects: `o_analysis`, `o_anaOption`, `o_envOption`, `o_simOption`, `o_keepOption`.
r_form	Form created by a call to `hiCreateAppForm` or `hiCreateForm`.
s_fieldName	Handle to this field within a form. It may be referenced as `formHandle->hiFieldSym`
g_value	The value you want to choose.

Values Returned

`t`	When the check is successful.
`nil`	Otherwise

asiFormatGraphicalStimuli

```
asiFormatGraphicalStimuli( o_session  p_fp )
     => t | nil
```

Description

Formats the graphical stimuli statements to send to Cadence SPICE.

> **Note:** For Integrators, the function `asiFormatGraphicalStimuli` needs to be overloaded for the simulator session to create a file which contains the graphical stimuli. If you define a graphical stimuli, it is mandatory for this to be overloaded , else an error stating that the function `asiFormatGraphicalStimuli` is not defined for the appropriate class, is displayed:
>
> `… *Error* asiFormatGraphicalStimuli: no applicable method for the class - <yoursimulator>_session`
>
> Also, the function `asiPrintSource` needs to be written by the integrator.

Arguments

o_session	Simulation session object. For example, an XYZ session is XYZ_session for a 3rd party.
p_fp	Specifies a file pointer to the file containing the graphical stimuli statements to send to the simulator (for simulators that are not in the Cadence SPICE socket).

Value Returned

`t`	Returns `t` if the stimuli statements are generated.
`nil`	Returns `nil` otherwise.

Example

The function prototype is as follows:

```
asiFormatGraphicalStimuli ((session XYZ_session) fp)
```

where, `fp` is the pointer to the file where graphical stimuli will be written.

asiFormatGraphicalStimulusFileList

```
asiFormatGraphicalStimulusFileList(
     o_session
     &_fp
     )
     => t_string
```

Description

Formats the statement that includes the graphical stimulus files.

Arguments

o_session	Simulation session object.
&_fp	Pointer to the control statement file.

Value Returned

t_string	Formatted string that includes the graphical stimulus file.

Example

```
defmethod ( asiFormatGraphicalStimulusFileList ( (session xyz_session) fp )
      (artFprintf fp "include \"%s\"\n" "./_graphical_stimuli.scs")
)
```

asiAddOceanAlias

```
asiAddOceanAlias( s_simulatorName s_alias s_analysisName )
     => t | nil
```

Description

Adds an ocean alias to the current simulator. This is useful for defining ocean related data access aliases for third party simulator integration.

Arguments

s_simulatorName	Simulator or Tool's class name
s_alias	Data access alias for specified analysis name
s_analysisName	Equivalent analysis name recognized by 3rd party simulator

Values Returned

`t`	When the alias is added and defined.
`nil`	Otherwise.

Example

For a 3rd party simulator integration if the simulator PSF maps transient analysis name as `analysisTran1-tran` then following OCEAN aliasing should be used:

```
asiAddOceanAlias( 'spice3 'tran "analysisTran1-tran")
```

This would make `selectResults('tran)` work in the auto generated OCEAN script file

asiCornerSimCB

```
asiCornerSimCB(o_session)
     => t | nil
```

Description

This function is called on successful completion of all the Corner runs. This is used to reorganize multi-run Corner simulation data corresponding to multiple Alter statements, in simulation input files.

Arguments

o_session	The session object.

Values Returned

t	Callback called.
nil	Callback not called.

Example

```
defmethod( asiCornerSimCB (( session  spice3_session ))
    (let ((retVal nil) ( cornerList reverse(asiGetCornerList( session ))) (psfPath
asiGetPsfDir(session)) corPath (cornIdx 2) copyList analList inPort outPort line )
when(and(cornerList asiGetEnvOptionVal(session 'useAltergroup))


;; copyList -> list of files that will be copied as-is
;; from 'psf' folder to Corners/cor_i folders.


    copyList = '("simRunData" "artistLogFile" "variables_file" "runObjFile")


;; Retain/move the parent 'runObjFile' file present in 'psf' folder
;; (generated by OASIS environment) to 'runObjFile.bak'


    errset(renameFile(strcat(psfPath "/runObjFile") strcat(psfPath "/
runObjFile.bak" )))


;; Generate the template 'runObjFile' (child) file in 'psf' folder.
;; This will be copied later to Corners/cor_i folders.
```

```
asiCreateRunObjectFile(session)
    foreach( corner cornerList
        corPath=strcat(psfPath "/../Corners/")
        corPath=strcat(corPath get_pname( asiGetCornerName( corner )))

;; Create the 'Corner' folder

    unless( isDir(corPath) createDir(corPath))
    foreach( file copyList

;; Copy the basic files as-is from ./psf to the Corners/cor_i folder

        system (sprintf nil "cp %s/%s %s" psfPath file corPath))

;; foreach Construct the relevant list of analysis data files for a Corner run
;; Integrator should change this according to the name of files
;; generated by their simulators.

sprintf(analList "analysisAC%d.ac analysisDC%d.dc analysisTran%d.tran" cornIdx
cornIdx cornIdx)

analList=parseString(analList)

;; Copy the analysis data files for that RUN into corresponding Corners/cor_i folder

    foreach( file setof( x getDirFiles(psfPath) member( x analList))
        errset(renameFile(strcat(psfPath "/" file) strcat(corPath "/" file))))

;; for each of them, process the ./psf/logFile and derive ./Corners/cor_i/logFile
for every corner RUN

    inPort=infile(strcat(psfPath "/logFile"))
    outPort=outfile(strcat(corPath "/logFile") "w")

;; Get the HEADER & TYPE fields as-is from the base logFile (./psf/logFile)

    while( and( inPort outPort gets(line inPort) setq(line parseString(line))
    foreach(mapcar i line fprintf(outPort "%s " i))
         newline(outPort)
        ) ;; while
        fprintf(outPort "VALUE\n")
```

```
;; Filter the Relevant analysis information for that Corner RUN.

    while( and( inPort outPort setq(line lineread(inPort)))
        if( member(nth(1 nth(2 line)) analList)
            then
            ;; Translate the analysis name dumped by simulator to somethin
            ;; common for the same analysis type across all Corners.
            ;; This is required for letting the Result reader know that they
            ;; all (different simulation data analysisAC2.ac, analysisAC3.ac
            ;; and so on) belong to the same family.
            ;; So change the name from 'analysisAC1-ac', 'analysisAC2-ac'
            ;; and so on to common name 'analysis-ac'.

        fprintf(outPort "'analysis-%s' %L " nth(0 nth(2 line)) nth(1 line))

            ;; Print Rest of the analysis data as is

        printlev(nth(2 line) 10 100 outPort)
            newline(outPort)
            ) ;; if
        ) ;; while

        fprintf(outPort "END")
        close(inPort)
        close(outPort)
        postincrement(cornIdx)
        ) ;; foreach

    ;; Delete all the files in the 'psf' folder.

    copyList = cons( "logFile" copyList)
    foreach( file setof( x getDirFiles(psfPath) member( x copyList))
        deleteFile(strcat(psfPath "/" file))
        ) ;; foreach

    ; Retain the original Parent 'runObjFile' file in 'psf' folder.

        errset(renameFile(strcat(psfPath "/runObjFile.bak") strcat(psfPath "/
runObjFile" )))
    retVal=t
    ) ;; when
```

```
retVal
)
)
```

asiGetCornerDesignVarList

```
asiGetCornerDesignVarList(o_corner)
     => l_cornerDesignVar
```

Description

This functions retrieves the list of design variables (*name value* pairs) defined for a Corner object. The *name* and *value* are of type string.

Arguments

o_corner	Corner object.

Values Returned

l_cornerDesignVar	List of design variables.

Example

The following SKILL code illustrates how you can use this function to extract Corner information:

```
myCornList = asiGetCornerList(session)
=> (Corner_1 Corner_2)
corn_1 = nth(0 myCornList)
=> Corner_1
corn_2 = nth(1 myCornList)
=> Corner_2
asiGetCornerName(corn_1)
=> "slowslow"
asiGetCornerName(corn_2)
=> "fastfast"
asiGetCornerTemperature(corn_2)
=> 125
asiGetCornerDesignVarList(corn_1)
=> (("C"  "10p")  ("R"  "10K"))
asiGetCornerModelSelectionList(corn_1)
=> (("File_1"  "slow") ("File_2"  "slow"))
asiGetCornerModelSelectionList(corn_2)
=> (("File_1"  "fast") ("File_2"  "fast"))
```

asiGetCornerList

```
asiGetCornerList(o_session)
     => l_corners
```

Description

This function returns the list of Corner objects (*o_corner*) defined for a session.

Arguments

o_session	The OASIS session object.

Values Returned

l_corners	List of Corner objects.

Example

The following SKILL code illustrates how you can use this function to extract Corner information:

```
myCornList = asiGetCornerList(session)
=> (Corner_1 Corner_2)
corn_1 = nth(0 myCornList)
=> Corner_1
corn_2 = nth(1 myCornList)
=> Corner_2
asiGetCornerName(corn_1)
=> "slowslow"
asiGetCornerName(corn_2)
=> "fastfast"
asiGetCornerTemperature(corn_2)
=> 125
asiGetCornerDesignVarList(corn_1)
=> (("C"  "10p")  ("R"  "10K"))
asiGetCornerModelSelectionList(corn_1)
=> (("File_1"  "slow") ("File_2"  "slow"))
asiGetCornerModelSelectionList(corn_2)
=> (("File_1"  "fast") ("File_2"  "fast"))
```

asiGetCornerModelSelectionList

```
asiGetCornerModelSelectionList(o_corner)
     => l_cornerModelFiles
```

Description

This function returns the list of Corner model files (*model_file, section* pairs) defined for a Corner object. The *model_file* and *section* are of type string.

Arguments

o_corner	Corner object.

Value Returned

l_cornerModelFiles	List of Corner model files.

Example

The following SKILL code illustrates how you can use this function to extract Corner information:

```
myCornList = asiGetCornerList(session)
=> (Corner_1 Corner_2)
corn_1 = nth(0 myCornList)
=> Corner_1
corn_2 = nth(1 myCornList)
=> Corner_2
asiGetCornerName(corn_1)
=> "slowslow"
asiGetCornerName(corn_2)
=> "fastfast"
asiGetCornerTemperature(corn_2)
=> 125
asiGetCornerDesignVarList(corn_1)
=> (("C"  "10p")  ("R"  "10K"))
asiGetCornerModelSelectionList(corn_1)
=> (("File_1"  "slow") ("File_2"  "slow"))
asiGetCornerModelSelectionList(corn_2)
=> (("File_1"  "fast") ("File_2"  "fast"))
```

asiGetCornerModelingStyles

```
asiGetCornerModelingStyles(o_session)
     => l_cornerModelingStyle
```

Description

This function returns the supported modeling styles for a simulator session.

Arguments

o_session	The OASIS session object

Values Returned

l_cornerModelingStyle	List of supported modeling styles

Example

```
defmethod( asiGetCornerModelingStyles (( session spice3_session))
   ;; Define these two modeling styles as the default ones for 3rd party Sims
   '("singleModelLib" "singleNumeric")
)
```

asiGetCornerName

```
asiGetCornerName(o_corner)
     => t_cornerName
```

Description

This function retrieves the name corresponding to a Corner object. The Corner name is of type string.

Arguments

o_corner	Corner object

Value Returned

t_cornerName	Corner name

Example

The following SKILL code illustrates how you can use this function to extract Corner information:

```
myCornList = asiGetCornerList(session)
=> (Corner_1 Corner_2)
corn_1 = nth(0 myCornList)
=> Corner_1
corn_2 = nth(1 myCornList)
=> Corner_2
asiGetCornerName(corn_1)
=> "slowslow"
asiGetCornerName(corn_2)
=> "fastfast"
asiGetCornerTemperature(corn_2)
=> 125
asiGetCornerDesignVarList(corn_1)
=> (("C"  "10p")  ("R"  "10K"))
asiGetCornerModelSelectionList(corn_1)
=> (("File_1"  "slow") ("File_2"  "slow"))
asiGetCornerModelSelectionList(corn_2)
=> (("File_1"  "fast") ("File_2"  "fast"))
```

asiGetCornerResultsDir

```
asiGetCornerResultsDir(o_corner)
     => t_cornerResDir
```

Description

This function retrieves the simulation result directory defined for a Corner object. The simulation result directory is of type string.

Arguments

o_corner	Corner object

Value Returned

t_cornerResDir	Simulation result directory

Example

The following SKILL code illustrates how you can use this function to extra Corner information:

```
myCornList = asiGetCornerList(session)
=> (Corner_1 Corner_2)
corn_1 = nth(0 myCornList)
=> Corner_1
corn_2 = nth(1 myCornList)
=> Corner_2
asiGetCornerName(corn_1)
=> "slowslow"
asiGetCornerName(corn_2)
=> "fastfast"
asiGetCornerTemperature(corn_2)
=> 125
asiGetCornerDesignVarList(corn_1)
=> (("C"  "10p")  ("R"  "10K"))
asiGetCornerModelSelectionList(corn_1)
=> (("File_1"  "slow") ("File_2"  "slow"))
asiGetCornerModelSelectionList(corn_2)
=> (("File_1"  "fast") ("File_2"  "fast"))
```

asiGetCornerTemperature

```
asiGetCornerTemperature(o_corner)
     => t_cornerTemp
```

Description

This function retrieves the simulation temperature defined for a Corner object. The simulation temperature is of type string.

Arguments

o_corner	Corner object

Value Returned

t_cornerTemp	Simulation temperature

Example

The following SKILL code illustrates how you can use this function to extract Corner information:

```
myCornList = asiGetCornerList(session)
=> (Corner_1 Corner_2)
corn_1 = nth(0 myCornList)
=> Corner_1
corn_2 = nth(1 myCornList)
=> Corner_2
asiGetCornerName(corn_1)
=> "slowslow"
asiGetCornerName(corn_2)
=> "fastfast"
asiGetCornerTemperature(corn_2)
=> 125
asiGetCornerDesignVarList(corn_1)
=> (("C"  "10p")  ("R"  "10K"))
asiGetCornerModelSelectionList(corn_1)
=> (("File_1"  "slow") ("File_2"  "slow"))
asiGetCornerModelSelectionList(corn_2)
=> (("File_1"  "fast") ("File_2"  "fast"))
```

OASIS Print Functions

artOutfile

```
artOutfile( t_name t_mode x_len t_break t_cont t_begCom t_endCom t_tab
     t_comments )
     => x_handle
```

Description

Opens the named file. The first argument is mandatory. The other arguments keep their default values if they are not set.

Arguments

t_name	Name of the file.
t_mode	Mode in which to open the file, either `w` or `a`. The default value is `w`.
x_len	Maximum number of chars before each new line. The default value is `internal default`.
t_break	Break characters, where breaking the line is legal. For example, "\t])". The default value is `null`.
t_cont	Continuation convention. For example, "\\\n" or "\n+". The default value is `null`.
t_begCom	Comment convention. For example, ";" or "slash-asterisk". The default value is `null`.
t_endCom	Complement of *beginComment*. For example, "\n" or asterisk-slash.
t_tab	Indicates if a tab should be replaced with a blank. The default value is `1`.
t_comments	Indicates if comments can be ignored. The default value is `1`.

Value Returned

x_handle	The named file.

Example

```
fHandle = artOutfile( "aFile" "a" )
```

artFprintf

```
artFprintf( x_handle t_text g_args ) => t | nil
```

Description

Prints out data like the standard C library `fprintf`, with the handle returned from `artOutfile()` as the first argument.

Arguments

x_handle	File handle returned from `artOutfile`.
t_text	Formatting string.
g_args	Data to be printed.

Value Returned

`t`	Returns `t` if the data is printed.
`nil`	Returns `nil` if there is an error.

Example

```
artFprintf( fHandle "%s" "test" )
```

artClose

```
artClose( x_handle ) => t | nil
```

Description

Closes the file associated with the given handle.

Arguments

x_handle	File handle returned from `artOutfile`.

Value Returned

`t`	Returns `t` if the file is closed.
`nil`	Returns `nil` if there is an error.

Example

```
artClose( fHandle )
```

artCloseAllFiles

```
artCloseAllFiles() => t | nil
```

Description

Closes all files opened with `artOutfile()`.

Value Returned

`t`	Returns `t` if the files are closed.
`nil`	Returns `nil` if there is an error.

Example

```
artCloseAllFiles()
```

artFlush

```
artFlush( x_handle ) => t | nil
```

Description

Flushes the file associated with the given handle.

Argument

x_handle	File handle returned from `artOutfile`.

Value Returned

`t`	Returns `t` if the file is flushed.
`nil`	Returns `nil` if there is an error.

Example

```
artFlush( fHandle )
```

artListOpenFiles

```
artListOpenFiles() => l_names | nil
```

Description

Lists names of all files opened with artOutfile()

Value Returned

`t`	Returns `t` if the file names are listed.
`nil`	Returns `nil` if there is an error.

Example

```
fNameList = artListOpenFiles()
```

7

Environment Variable Functions

This chapter describes functions that let you work with environment variables.

asiAddEnvOption

```
asiAddEnvOption(
     o_tool
     ?name          s_name
     [?prompt       t_prompt]
     [?type         s_type]
     [?choices      l_choices]
     [?itemsPerRow  x_itemsPerRow]
     [?value        g_value]
     [?min          g_min]
     [?max          g_max]
     [?allowExpr    s_allowExpr]
     [?row          x_row]
     [?column       x_column]
     [?width        x_width]
     [?coordinates  l_coordinates]
     [?displayOrder x_displayOrder]
     [?labelText    t_labelText]
     [?private      s_private]
     [?display      s_display]
     [?editable     s_editable]
     [?appCB        s_appCB]
     [?callback     t_callback]
     [?formApplyCB  s_formApplyCB]
     [?changeCB     st_changeCB]
     [?doubleClickCB   st_doubleClickCB]
     [?numRows      x_numRows]
     [?multipleSelect   s_multipleSelect]
     [?browse       g_browse]
     [?mode         t_browseMode]
     [?invalidateFunc   s_invalidateFunc]
     [?defaultSubcircuitCall  s_defaultSubcircuitCall]
     )
     => o_envVar | nil
```

Description

Adds a new simulation environment option.

Arguments

o_tool	Simulation tool object.
s_name	Name of the environment option you want to add.

t_prompt	Optional argument that specifies the prompt (on the UI form) for the given option. Default Value: *s_name*
s_type	Type of the option. Valid Values: `string`, `integer`, `float`, `toggle`, `cyclic`, `radio`, `boolean`, `list`, `radioToggle`, `listBox`, `fileName` (string type for file names only), `label` (for the label on the UI form), `frame` (for a graphic frame around a field), `separator` (for the separator line on the UI form), `button` (for a button on the UI form), `scale` (for a slider field on the UI form) Default Value: `string` See the asiAddSimOption function in the "Simulator Option Functions" chapter for an example that shows how to use the *separator* argument.
l_choices	List of choices if *s_type* is `cyclic`, `radio`, `radioToggle` or `listBox`, or the list of switches if *s_type* is `toggle`. This argument is valid only if *s_type* is `cyclic`, `toggle`, `radio`, `radioToggle`, or `listBox`.
x_itemsPerRow	Numbers of choices per row for `radio`, `cyclic`, `toggle`, and `radioToggle` fields. Default Value: Total number of choices specified in *l_choices*
g_value	Default value of the option.
g_min	Specifies the minimum value of an `integer`, `float`, or `scale` option. Default Value: `nil`, which means `-infinity`
g_max	Specifies the maximum value of an `integer`, `float`, or `scale` option. Default Value: `nil`, which means `+infinity`
s_allowExpr	Specifies whether *g_value* can contain expressions. Valid Values: `t` (value can contain expressions), `nil` (value cannot contain expressions) Default Value: `nil`

x_row	Row in the form where the field appears. This argument is valid only for *'twoD* type forms.
x_column	Column in the form where the field appears. The fields are created according to their required widths and are not meant to align with fields in other rows of the form. This argument is valid only for *'twoD* type forms.
x_width	Specifies the width of the field in relation to other fields on the form. Numbers that you enter for this argument are relative values whose actual values are determined by the amount of space available. For example, assume there are three fields for a row that is 400 pixels wide. If the first two fields have an *x_width* of 1, and the last field has an *x_width* of 2, then the widths for the fields are as follows: 100, 100, and 200 pixels. This argument is valid only for *'twoD* type forms. Default Value: 1
l_coordinates	List specifying the coordinates for the field on the UI form. (The format is the same as for the corresponding *hi* field.) This argument is valid only for *'custom* type forms.
x_displayOrder	Position (from the top) of the option in the form. Use *x_displayOrder* to reposition your options in an *inherited* form. By default, the options appear in the order defined; therefore, inherited options appear first. The value of *x_displayOrder* must be an integer that specifies the position you want relative to the top of the form. If more than one option has the same display order integer, the last option found (yours) takes precedence. The remaining options shift down on the form. This argument is valid only for *'oneD* type forms. Valid Values: Any integer
t_labelText	Optional label displayed with frame type fields. Default Value: `nil`
s_private	Optional argument that declares an option as private, which means the option is not visible to the user. Private options are not part of the UI and their values are not saved when a user saves

the environment. You might use this argument for values that are constant for all users of the software. Valid Values: `t` (option does not appear in the UI), `nil` (option appears in the UI)
Default Value: `nil`

s_display — Specifies an expression that determines whether the field is to be displayed on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

s_editable — Specifies an expression that determines whether the field is to be editable on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

Note: This argument only applies to type-in fields.

s_appCB — Specifies a callback function that is executed when the value of the option is changed.
Callback parameter list: (*o_session*)

t_callback — Specifies a callback function that is executed when a field is changed on the form while the form is displayed. (You can use this function to do error checking on the value entered by the designer.)

s_formApplyCB — Specifies a callback function that is executed when the designer clicks on *Apply* or *OK* on the form containing the associated field.
Callback parameter list: (*o_session r_form r_field*)

st_changeCB — Specifies a callback function that is executed when the value of a `listBox` type field is changed on the form.

st_doubleClickCB — Specifies a callback function that is executed when a designer double clicks on a `listBox` type field.

x_numRows — Number of rows shown on the form for a `listBox` type field.

s_multipleSelect

Boolean flag that specifies whether multiple items can be selected from the `listBox` type field.
Valid Values: `t` (multiple items can be selected), `nil` (only one item can be selected at a time)

g_browse

Adds a browse button (...) for the specified option. This button can be used to open a file selection form to browse and select a file. The mode of file selection is specified by the *t_mode* argument.

Valid values: `t`, `nil`
Default value: `nil`

t_mode

Specifies mode for the file selection form that is displayed when the *g_browse* argument is set to `t`.

Valid values: `anyFile` specifies that you can open any file type; `existingFile` specifies that you can open any existing file; and `existingFiles` specifies that you can select multiple files.
Default value: `anyFile`

s_invalidateFunc

Specifies a function that is executed when the value of the option is changed. This function invalidates a step in the flowchart.
Callback parameter list: (*o_session*)

s_defaultSubcircuitCall

Specifies the name of the netlist procedure added to the cdf simInfo for sub-circuits that is added by the cell-view to cell-view utility. For interfaces derived from the asiAnalog class this is an empty string, and no netlist procedure is added. For interfaces derived from the asiSocket class this is "ansSpiceSubcktCall". To change the value, simply use asiChangeEnvOption in your envOption.il file.

Value Returned

o_envVar

Returns the environment option object.

nil

Returns `nil` if there is an error.

Example

```
asiAddEnvOption( tool ?name 'XYZfile ?prompt "XYZ file" ?value
"~/XYZfile/XYZfile" ?displayOrder 4 )
```

Adds a new environment option called XYZ file in the fourth position in the Environment Options form.

Related Functions

To display the current set of environment options, see the asiDisplayEnvOption function on page 7-258.

To change the display characteristics of your Environment Options form, see the asiChangeEnvOptionFormProperties function on page 7-255.

asiChangeEnvOption

```
asiChangeEnvOption(
     o_tool
     ?name          s_name
     [?prompt       t_prompt]
     [?type         s_type]
     [?choices      l_choices]
     [?itemsPerRow  x_itemsPerRow]
     [?value        g_value]
     [?min          g_min]
     [?max          g_max]
     [?allowExpr    s_allowExpr]
     [?row          x_row]
     [?column       x_column]
     [?width        x_width]
     [?coordinates  l_coordinates]
     [?displayOrder  x_displayOrder]
     [?labelText    t_labelText]
     [?private      s_private]
     [?display      s_display]
     [?editable     s_editable]
     [?appCB        s_appCB]
     [?callback     t_callback]
     [?formApplyCB  s_formApplyCB]
     [?changeCB     st_changeCB]
     [?doubleClickCB   st_doubleClickCB]
     [?numRows      x_numRows]
     [?multipleSelect  s_multipleSelect]
     [?invalidateFunc  s_invalidateFunc]
     )
     => o_envVar | nil
```

Description

Changes a simulation environment option.

Arguments

o_tool	Simulation tool object.
s_name	Name of the environment option you want to change.
t_prompt	Optional argument that specifies the prompt (on the UI form) for the given option. Default Value: *s_name*

s_type
Type of the option.
Valid Values: `string`, `integer`, `float`, `toggle`, `cyclic`, `radio`, `boolean`, `list`, `radioToggle`, `listBox`, `fileName` (string type for file names only), `label` (for the label on the UI form), `frame` (for a graphic frame around a field), `separator` (for the separator line on the UI form), `button` (for a button on the UI form), `scale` (for a slider field on the UI form)
Default Value: `string`

Note: See the asiAddSimOption function in the "Simulator Option Functions" chapter for an example that shows how to use the *separator* argument.

l_choices
List of choices if *s_type* is `cyclic`, `radio`, `radioToggle` or `listBox`, or the list of switches if *s_type* is `toggle`. This argument is valid only if *s_type* is `cyclic`, `toggle`, `radio`, `radioToggle`, or `listBox`.

x_itemsPerRow
Numbers of choices per row for `radio`, `cyclic`, `toggle`, and `radioToggle` fields.
Default Value: Total number of choices specified in *l_choices*

g_value
Default value of the option.

g_min
Specifies the minimum value of an `integer`, `float`, or `scale` option.
Default Value: `nil,` which means `-infinity`

g_max
Specifies the maximum value of an `integer`, `float`, or `scale` option.
Default Value: `nil`, which means `+infinity`

s_allowExpr
Specifies whether *g_value* can contain expressions.
Valid Values: `t` (value can contain expressions), `nil` (value cannot contain expressions)
Default Value: `nil`

x_row
Row in the form where the field appears. This argument is valid only for *'twoD* type forms.

x_column
Column in the form where the field appears. The fields are created according to their required widths and are not meant to align with fields in other rows of the form. This argument is valid only for *'twoD* type forms.

x_width

Specifies the width of the field in relation to other fields on the form. Numbers that you enter for this argument are relative values whose actual values are determined by the amount of space available. For example, assume there are three fields for a row that is 400 pixels wide. If the first two fields have an *x_width* of 1, and the last field has an *x_width* of 2, then the widths for the fields are as follows: 100, 100, and 200 pixels. This argument is valid only for *'twoD* type forms.
Default Value: 1

l_coordinates

List specifying the coordinates for the field on the UI form. (The format is the same as for the corresponding *hi* field.) This argument is valid only for *'custom* type forms.

x_displayOrder

Position (from the top) of the option in the form. Use *x_displayOrder* to reposition your options in an *inherited* form. By default, the options appear in the order defined; therefore, inherited options appear first.

The value of *x_displayOrder* must be an integer that specifies the position you want relative to the top of the form. If more than one option has the same display order integer, the last option found (yours) takes precedence. The remaining options shift down on the form. This argument is valid only for *'oneD* type forms.
Valid Values: Any integer

t_labelText

Optional label displayed with frame type fields.
Default Value: `nil`

s_private

Optional argument that declares an option as private, which means the option is not visible to the user. Private options are not part of the UI and their values are not saved when a user saves the environment. You might use this argument for values that are constant for all users of the software. Valid Values: `t` (option does not appear in the UI), `nil` (option appears in the UI)
Default Value: `nil`

s_display

Specifies an expression that determines whether the field is to be displayed on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on

any field in the form.
Default Value: `t`

s_editable Specifies an expression that determines whether the field is to be editable on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

Note: This argument only applies to type-in fields.

s_appCB Specifies a callback function that is executed when the value of the option is changed.
Callback parameter list: (*o_session*)

t_callback Specifies a callback function that is executed when a field is changed on the form while the form is displayed. (You can use this function to do error checking on the value entered by the designer.)

s_formApplyCB
Specifies a callback function that is executed when the designer clicks on *Apply* or *OK* on the form containing the associated field.
Callback parameter list: (*o_session r_form r_field*)

st_changeCB Specifies a callback function that is executed when the value of a `listBox` type field is changed on the form.

st_doubleClickCB
Specifies a callback function that is executed when a designer double clicks on a `listBox` type field.

x_numRows Number of rows shown on the form for a `listBox` type field.

s_multipleSelect
Boolean flag that specifies whether multiple items can be selected from the `listBox` type field.
Valid Values: `t` (multiple items can be selected), `nil` (only one item can be selected at a time)

s_invalidateFunc
Specifies a function that is executed when the value of the option

is changed. This function invalidates a step in the flowchart.
Callback parameter list: (*o_session*)

Value Returned

o_envVar	Returns the changed environment option object.
nil	Returns an error message and nil if unsuccessful.

Example

```
asiChangeEnvOption( tool ?name 'switchViewList
?value '("XYZ" "spice" "cmos.sch" "schematic" "symbol"))
```

Changes the default value of the switch view list.

Related Functions

To display the current set of environment options, see the asiDisplayEnvOption function on page 7-258.

To change the display characteristics of your Environment Options form, see the asiChangeEnvOptionFormProperties function on page 7-255.

asiChangeEnvOptionFormProperties

```
asiChangeEnvOptionFormProperties(
     o_tool
     [?type    s_type]
     [?width   x_width]
     [?columns x_columns]
     )
     => o_formObj | nil
```

Description

Changes the display characteristics of the Environment Options form.

Arguments

o_tool	Simulation tool object.
s_type	Specifies the form type. Valid Values:
'oneD	Specifies a sequential display of fields in one column.
'twoD	Specifies a two dimensional display of fields based on row and column positions. (Use the asiAddEnvOption or asiChangeEnvOption function to specify values for the rows and columns.)
'custom	Lets you specify the exact coordinate locations for each field. (Use the asiAddEnvOption or asiChangeEnvOption function to specify the coordinates.)
'matrix	Specifies a matrix of equally sized fields.
	Default Value: 'oneD
x_width	Width of the form, in pixels. Default Value: The minimum default width of the form is 400 pixels. If the fields require more space than this, the form defaults to the smallest width that can accommodate the fields.

x_columns	Number of columns. Use this argument only with matrix type forms. Default Value: 2

Value Returned

o_formObj	Returns the environment option form object if successful.
nil	Returns nil otherwise.

Example

```
asiChangeEnvOptionFormProperties( asiGetTool('spectreS)
?width 450)
```

Changes the width of the Environment Options form for the Spectre simulator.

asiDeleteEnvOption

```
asiDeleteEnvOption( o_tool s_name )
     => t | nil
```

Description

Deletes a simulation environment option.

Arguments

o_tool	Simulation tool object.
s_name	Name of the environment option you want to remove from the form.

Value Returned

`t`	Returns `t` when the option is deleted.
`nil`	Returns an error message and `nil` if the option does not exist.

Example

```
asiDeleteEnvOption( tool 'stimulusFile )
```

Removes the *stimulusFile* option from the Environment Options form.

Related Function

To display the current set of environment options, see the asiDisplayEnvOption function on page 7-258.

asiDisplayEnvOption

```
asiDisplayEnvOption( o_tool )
     => t | nil
```

Description

Displays the current set of simulation environment option names and values. Use this function only to determine which environment options you want to modify. Do not use this function as part of another procedure.

Arguments

o_tool	Simulation tool object.

Value Returned

`t`	Displays the list of environment option names and values and returns `t`.
`nil`	Returns `nil` otherwise.

Example

```
asiDisplayEnvOption( asiGetTool( 'spectreS ) )
```

Displays the *spectreS* environment options.

asiDisplayEnvOptionFormProperties

```
asiDisplayEnvOptionFormProperties( o_tool )
     => t | nil
```

Description

Displays the form characteristics for the Environment Options form. Use this function only to determine which form characteristics you want to modify. Do not use this function as part of another procedure.

Arguments

o_tool	Simulation tool object.

Value Returned

`t`	Displays a list of Environment Option form characteristics and returns `t`.
`nil`	Returns `nil` otherwise.

Example

```
asiDisplayEnvOptionFormProperties( asiGetTool('spectreS))
```

Displays the form characteristics for the Spectre Environment Options form and returns `t`.

asiGetEnvOptionChoices

```
asiGetEnvOptionChoices(
     {o_session | o_tool}
     s_name
     )
     => l_choices | nil
```

Description

Gets the list of choices for an environment option that is set up as a list box.

Arguments

o_session	Simulation session object.
o_tool	Simulation tool object.
s_name	Name of the option for which you want the list of choices.

Value Returned

l_choices	Returns the list of choices for the specified simulation environment option.
nil	Returns nil if the option does not exist.

Related Function

To display the current set of environment options, see the asiDisplayEnvOption function on page 7-258.

asiGetEnvOptionVal

```
asiGetEnvOptionVal(
     {o_session | o_tool}
     s_name
     )
     => g_value | nil
```

Description

Gets the value for the specified simulation environment option.

Arguments

o_session	Simulation session object.
o_tool	Simulation tool object.
s_name	Name of the option for which you want the value.

Value Returned

g_value	Returns the value for the specified simulation environment option.
nil	Returns `nil` if the option does not exist.

Example

```
asiGetEnvOptionVal( session 'modelPath )
```

Gets the value for the *modelPath* simulation environment option.

Related Function

To display the current set of environment options, see the asiDisplayEnvOption function on page 7-258.

asiInit<yourSimulator>EnvOption

```
asiInit<yourSimulator>EnvOption( o_tool )
     => t
```

Description

Calls your procedures to modify the Virtuoso analog design environment simulation options.

Note: You must write `asiInit<yourSimulator>EnvOption`, where <*yourSimulator*> is the name of your simulator. Do not include the angle brackets (<>).

Arguments

o_tool	Simulation tool object.

Value Returned

`t`	Returns `t` when the procedures are called.

Note: You must write `asiInit<yourSimulator>EnvOption` to return `t`.

Example

```
procedure( asiInitXYZEnvOption( tool )
    <insert your code>
    t
)
```

Creates the procedure that calls the procedures to add environment options for the XYZ simulator.

asiSetEnvOptionChoices

```
asiSetEnvOptionChoices(
     {o_session | o_tool}
     s_name
     l_choices
     )
     => l_choices | nil
```

Description

Specifies the list of choices to appear in the list box field for the specified environment option.

Arguments

o_session	Simulation session object.
o_tool	Simulation tool object.
s_name	Name a simulation environment option of the type `listBox`.
l_choices	List of choices to appear in the list box field.

Value Returned

l_choices	Returns the new list of choices to appear in the list box field.
`nil`	Returns `nil` if the option does not exist.

asiSetEnvOptionVal

```
asiSetEnvOptionVal(
     {o_session | o_tool}
     s_name
     g_value
     )
     => g_value | nil
```

Description

Sets the value of the specified simulation environment option.

Arguments

o_session	Simulation session object.
o_tool	Simulation tool object.
s_name	Name of the simulation environment option to which you want to assign a value.
g_value	Value for the simulation option.

Value Returned

g_value	Returns the new value for the simulation environment option.
`nil`	Returns an error message and `nil` if the option does not exist.

Example

```
asiSetEnvOptionVal(session "modelFiles" '("/mypath/mySpectreModels.scs"))
```

Sets the value of modelFiles to /mypath/mySpectreModels.scs for Spectre Direct.

```
asiSetEnvOptionVal(session "modelFiles" '(("/mypath/mySpectreModels.scs"
"cmos")))
```

Sets the value of modelFiles to /mypath/mySpectreModels.scs and the section name to cmos for Spectre Direct.

```
asiSetEnvOptionVal( session 'modelPath "~models/nmos" )
```

Sets the value of the *modelPath* simulation environment option to `~models/nmos` for spectreS.

Related Function

To display the current set of environment options, see the asiDisplayEnvOption function on page 7-258.

8

Simulator Option Functions

This chapter describes functions that let you modify your simulator-specific options.

asiAddSimOption

```
asiAddSimOption(
     o_tool
     ?name          s_name
     [?prompt       t_prompt]
     [?type         s_type]
     [?choices      l_choices]
     [?itemsPerRow  x_itemsPerRow]
     [?value        g_value]
     [?min          g_min]
     [?max          g_max]
     [?allowExpr    s_allowExpr]
     [?row          x_row]
     [?column       x_column]
     [?width        x_width]
     [?coordinates  l_coordinates]
     [?displayOrder x_displayOrder]
     [?labelText    t_labelText]
     [?private      s_private]
     [?display      s_display]
     [?editable     s_editable]
     [?appCB        s_appCB]
     [?callback     t_callback]
     [?formApplyCB  s_formApplyCB]
     [?changeCB     st_changeCB]
     [?doubleClickCB   st_doubleClickCB]
     [?numRows      x_numRows]
     [?multipleSelect  s_multipleSelect]
     [?browse       g_browse]
     [?mode         t_browseMode]
     [?invalidateFunc  s_invalidateFunc]
     [?genericName  s_genericName]
     [?sendMethod   s_sendMethod]
     )
     => o_envVar | nil
```

Description

Adds a new simulator option.

Arguments

o_tool	Simulation tool object.
s_name	Name of the simulator option you want to define.

t_prompt Optional argument that specifies the prompt (on the UI form) for the given option.
Default Value: *s_name*

s_type Type of the option.
Valid Values: `string, integer, float, toggle, cyclic, radio, boolean, list, radioToggle, listBox, fileName` (string type for file names only), `label` (for the label on the UI form), `frame` (for a graphic frame around a field), `separator` (for the separator line on the UI form), `button` (for a button on the UI form), `scale` (for a slider field on the UI form)
Default Value: `string`

Note: The example that follows shows how to use the separator argument.

l_choices List of choices if *s_type* is `cyclic, radio, radioToggle` or `listBox`, or the list of switches if *s_type* is `toggle`. This argument is valid only if *s_type* is `cyclic, toggle, radio, radioToggle`, or `listBox`.

x_itemsPerRow
Numbers of choices per row for `radio, cyclic, toggle,` and `radioToggle` fields.
Default Value: Total number of choices specified in *l_choices*

g_value Default value of the option.

g_min Specifies the minimum value of an `integer, float,` or `scale` option.
Default Value: `nil`, which means `-infinity`

g_max Specifies the maximum value of an `integer`, `float`, or `scale` option.
Default Value: `nil`, which means `+infinity`

s_allowExpr Specifies whether *g_value* can contain expressions.
Valid Values: `t` (value can contain expressions), `nil` (value cannot contain expressions)
Default Value: `nil`

x_row Row in the form where the field appears. This argument is valid only for *'twoD* type forms.

x_column	Column in the form where the field appears. The fields are created according to their required widths and are not meant to align with fields in other rows of the form. This argument is valid only for *'twoD* type forms.
x_width	Specifies the width of the field in relation to other fields on the form. Numbers that you enter for this argument are relative values whose actual values are determined by the amount of space available. For example, assume there are three fields for a row that is 400 pixels wide. If the first two fields have an *x_width* of 1, and the last field has an *x_width* of 2, then the widths for the fields are as follows: 100, 100, and 200 pixels. This argument is valid only for *'twoD* type forms. Default Value: 1
l_coordinates	List specifying the coordinates for the field on the UI form. (The format is the same as for the corresponding *hi* field.) This argument is valid only for *'custom* type forms.
x_displayOrder	Position (from the top) of the option in the form. Use *x_displayOrder* to reposition your options in an *inherited* form. By default, the options appear in the order defined; therefore, inherited options appear first. The value of *x_displayOrder* must be an integer that specifies the position you want relative to the top of the form. If more than one option has the same display order integer, the last option found (yours) takes precedence. The remaining options shift down on the form. This argument is valid only for *'oneD* type forms. Valid Values: Any integer
t_labelText	Optional label displayed with frame type fields. Default Value: `nil`
s_private	Optional argument that declares an option as private, which means the option is not visible to the user. Private options are not part of the UI and their values are not saved when a user saves the environment. You might use this argument for values that are constant for all users of the software. Valid Values: `t` (option does

not appear in the UI), `nil` (option appears in the UI)
Default Value: `nil`

s_display Specifies an expression that determines whether the field is to be displayed on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

s_editable Specifies an expression that determines whether the field is to be editable on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

Note: This argument only applies to type-in fields.

s_appCB Specifies a callback function that is executed when the value of the option is changed.
Callback parameter list: (*o_session*)

t_callback Specifies a callback function that is executed when a field is changed on the form while the form is displayed. (You can use this function to do error checking on the value entered by the designer.)

s_formApplyCB Specifies a callback function that is executed when the designer clicks on *Apply* or *OK* on the form containing the associated field.
Callback parameter list: (*o_session r_form r_field*)

st_changeCB Specifies a callback function that is executed when the value of a `listBox` type field is changed on the form.

st_doubleClickCB Specifies a callback function that is executed when a designer double clicks on a `listBox` type field.

x_numRows Number of rows shown on the form for a `listBox` type field.

s_multipleSelect Boolean flag that specifies whether multiple items can be selected from the `listBox` type field.

Valid Values: `t` (multiple items can be selected), `nil` (only one item can be selected at a time)

g_browse — Adds a browse button (...) for the specified option. This button can be used to open a file selection form to browse and select a file. The mode of file selection is specified by the *t_mode* argument.

Valid values: `t`, `nil`
Default value: `nil`

t_mode — Specifies mode for the file selection form that is displayed when the *g_browse* argument is set to `t`.

Valid values: `anyFile` specifies that you can open any file type; `existingFile` specifies that you can open any existing file; and `existingFiles` specifies that you can select multiple files.
Default value: `anyFile`

s_invalidateFunc

Specifies a function that is executed when the value of the option is changed. This function invalidates a step in the flowchart.
Callback parameter list: (*o_session*)

s_genericName

Specifies the name of the generic variable that the option represents. This argument applies to temperature options only.
Valid Values: `temperature` specifies that the option represents the temperature for the simulator, `nominalTemp` specifies that the option represents the nominal temperature for the simulator

s_sendMethod — Specifies how simulator options are sent to Cadence SPICE. (Use this argument only if your simulator is in the SPICE Socket.)
Valid Values:

`set`	Specifies options known by Cadence SPICE (for example, `tempdc` and `dcoppt`)
`ptprop`	Specifies numbers
`psprop`	Specifies strings

Default Value: `ptprop`

Note: You can specify other values for the sendMethod argument. However, options must be sent to Cadence SPICE with the appropriate set, ptprop, or psprop statements.

Value Returned

o_envVar	Returns the simulator option object.
nil	Returns nil if there is an error.

Example

```
asiAddSimOption( tool ?name 'gmin ?value "1e-12" )
```

Adds the gmin simulator option, which is set to the value of *1e-12*.

```
procedure( asiInit<yourSimulator>SimOption( tool )

;*************************************************************
; TOLERANCE OPTIONS
;*************************************************
asiAddSimOption( tool
    ?name           'separator1
    ?type           'separator
)
asiAddSimOption( tool
    ?name           'label1
    ?type           'label
    ?prompt         "TOLERANCE OPTIONS"
)
asiAddSimOption( tool
    ?name           'reltol
    ?type           'string
    ?value          "1e-3"
)
asiAddSimOption( tool
    ?name           'conv
    ?type           'cyclic
    ?value          "off"
    ?choices        '( "off" "on" )
    ?sendMethod          'psprop
)

;*************************************************************
; TEMPERATURE OPTIONS
;*************************************************


asiAddSimOption( tool
    ?name           'separator2
    ?type           'separator
)
```

```
asiAddSimOption( tool
    ?name           'label2
    ?type           'label
    ?prompt         "TEMPERATURE OPTIONS"
)
asiAddSimOption( tool
    ?name           'temp
    ?type           'string
    ?value          "27"
    ?genericName    'temperature
)
```

Adds the `reltol, conv`, and `temp` options. Separator lines are drawn between these sections on the form.

Note: This example is for a 'oneD form.

Related Functions

To display the current set of simulator options, see the asiDisplaySimOption function on page 8-284.

To change the display characteristics for your Simulator Options form, see the asiChangeSimOptionFormProperties function on page 8-281.

asiChangeSimOption

```
asiChangeSimOption(
     o_tool
     ?name          s_name
     [?prompt       t_prompt]
     [?type         s_type]
     [?choices      l_choices]
     [?itemsPerRow  x_itemsPerRow]
     [?value        g_value]
     [?min          g_min]
     [?max          g_max]
     [?allowExpr    s_allowExpr]
     [?row          x_row]
     [?column       x_column]
     [?width        x_width]
     [?coordinates  l_coordinates]
     [?displayOrder x_displayOrder]
     [?labelText    t_labelText]
     [?private      s_private]
     [?display      s_display]
     [?editable     s_editable]
     [?appCB        s_appCB]
     [?callback     t_callback]
     [?formApplyCB  s_formApplyCB]
     [?changeCB     st_changeCB]
     [?doubleClickCB   st_doubleClickCB]
     [?numRows      x_numRows]
     [?multipleSelect  s_multipleSelect]
     [?invalidateFunc  s_invalidateFunc]
     [?genericName  s_genericName]
     [?sendMethod   s_sendMethod]
     )
     => o_envVar | nil
```

Description

Changes a simulator option.

Arguments

o_tool	Simulation tool object.
s_name	Name of the simulator option you want to change.

t_prompt Optional argument that specifies the prompt (on the UI form) for the given option.
Default Value: *s_name*

s_type Type of the option.
Valid Values: `string, integer, float, toggle, cyclic, radio, boolean, list, radioToggle, listBox, fileName` (string type for file names only), `label` (for the label on the UI form), `frame` (for a graphic frame around a field), `separator` (for the separator line on the UI form), `button` (for a button on the UI form), `scale` (for a slider field on the UI form)
Default Value: `string`

Note: See the asiAddSimOption function for an example that shows how to use the *separator* argument.

l_choices List of choices if *s_type* is `cyclic`, `radio`, `radioToggle` or `listBox`, or the list of switches if *s_type* is `toggle`. This argument is valid only if *s_type* is `cyclic`, `toggle`, `radio`, `radioToggle`, or `listBox`.

x_itemsPerRow
Numbers of choices per row for `radio, cyclic, toggle,` and `radioToggle` fields.
Default Value: Total number of choices specified in *l_choices*

g_value Default value of the option.

g_min Specifies the minimum value of an `integer`, `float`, or `scale` option.
Default Value: `nil`, which means `-infinity`

g_max Specifies the maximum value of an `integer`, `float`, or `scale` option.
Default Value: `nil`, which means `+infinity`

s_allowExpr Specifies whether *g_value* can contain expressions.
Valid Values: `t` (value can contain expressions), `nil` (value cannot contain expressions)
Default Value: `nil`

x_row Row in the form where the field appears. This argument is valid only for 'twoD type forms.

x_column	Column in the form where the field appears. The fields are created according to their required widths and are not meant to align with fields in other rows of the form. This argument is valid only for 'twoD type forms.
x_width	Specifies the width of the field in relation to other fields on the form. Numbers that you enter for this argument are relative values whose actual values are determined by the amount of space available. For example, assume there are three fields for a row that is 400 pixels wide. If the first two fields have an *x_width* of 1, and the last field has an *x_width* of 2, then the widths for the fields are as follows: 100, 100, and 200 pixels. This argument is valid only for *'twoD* type forms. Default Value: 1
l_coordinates	List specifying the coordinates for the field on the UI form. (The format is the same as for the corresponding *hi* field.) This argument is valid only for 'custom type forms.
x_displayOrder	Position (from the top) of the option in the form. Use *x_displayOrder* to reposition your options in an *inherited* form. By default, the options appear in the order defined; therefore, inherited options appear first. The value of *x_displayOrder* must be an integer that specifies the position you want relative to the top of the form. If more than one option has the same display order integer, the last option found (yours) takes precedence. The remaining options shift down on the form. This argument is valid only for 'oneD type forms. Valid Values: Any integer
t_labelText	Optional label displayed with frame type fields. Default Value: `nil`
s_private	Optional argument that declares an option as private, which means the option is not visible to the user. Private options are not part of the UI and their values are not saved when a user saves the environment. You might use this argument for values that are constant for all users of the software. Valid Values: `t` (option does

not appear in the UI), `nil` (option appears in the UI)
Default Value: `nil`

s_display — Specifies an expression that determines whether the field is to be displayed on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

s_editable — Specifies an expression that determines whether the field is to be editable on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

Note: This argument only applies to type-in fields.

s_appCB — Specifies a callback function that is executed when the value of the option is changed.
Callback parameter list: (*o_session*)

t_callback — Specifies a callback function that is executed when a field is changed on the form while the form is displayed. (You can use this function to do error checking on the value entered by the designer.)

s_formApplyCB — Specifies a callback function that is executed when the designer clicks on *Apply* or *OK* on the form containing the associated field.
Callback parameter list: (*o_session r_form r_field*)

st_changeCB — Specifies a callback function that is executed when the value of a `listBox` type field is changed on the form.

st_doubleClickCB — Specifies a callback function that is executed when a designer double clicks on a `listBox` type field.

x_numRows — Number of rows shown on the form for a `listBox` type field.

s_multipleSelect — Boolean flag that specifies whether multiple items can be selected from the *listBox* type field.

Valid Values: `t` (multiple items can be selected), `nil` (only one item can be selected at a time)

s_invalidateFunc

Specifies a function that is executed when the value of the option is changed. This function invalidates a step in the flowchart. Callback parameter list: (*o_session*)

s_genericName

Specifies the name of the generic variable that the option represents. This argument applies to temperature options only. Valid Values: `temperature` specifies that the option represents the temperature for the simulator, `nominalTemp` specifies that the option represents the nominal temperature for the simulator

s_sendMethod Specifies how simulator options are sent to Cadence SPICE. (Use this argument only if your simulator is in the SPICE Socket.) Valid Values:

`set`	Specifies options known by Cadence SPICE (for example, `tempdc` and `dcoppt`).
`ptprop`	Specifies numbers.
`psprop`	Specifies strings.

Default Value: `ptprop`

Note: You can specify other values for the `sendMethod` argument. However, options must be sent to Cadence SPICE with the appropriate `set`, `ptprop`, or `psprop` statements.

Value Returned

o_envVar	Returns the changed simulator option object.
`nil`	Returns an error message and `nil` if unsuccessful.

Related Functions

To display the current set of simulator options, see the asiDisplaySimOption function on page 8-284.

To change the display characteristics for your Simulator Options form, see the asiChangeSimOptionFormProperties function on page 8-281.

asiChangeSimOptionFormProperties

```
asiChangeSimOptionFormProperties(
     o_tool
     [?type    s_type]
     [?width   x_width]
     [?columns x_columns]
     )
     => o_formObj | nil
```

Description

Changes the display characteristics for the Simulator Options form.

Arguments

o_tool	Simulation tool object.
s_type	Specifies the form type. Valid Values:
'oneD	Specifies a sequential display of fields in one column.
'twoD	Specifies a two dimensional display of fields based on row and column positions. (Use the asiAddSimOption or asiChangeSimOption function to specify values for the rows and columns.)
'custom	Lets you specify the exact coordinate locations for each field. (Use the asiAddSimOption or asiChangeSimOption function to specify the coordinates.)
'matrix	Specifies a matrix of equally sized fields.
	Default Value: 'oneD
x_width	Width of the form, in pixels. Default Value: The minimum default width of the form is 400 pixels. If the fields require more space than this, the form defaults to the smallest width that can accommodate the fields.

x_columns	Number of columns. Use this argument only with matrix type forms. Default Value: 2

Value Returned

o_formObj	Returns the simulator option form object if successful.
nil	Returns nil otherwise.

Example

```
asiChangeSimOptionFormProperties( asiGetTool('spectreS)
    ?width 450 )
```

Changes the width of the Simulator Options form for the Spectre simulator.

asiDeleteSimOption

```
asiDeleteSimOption( o_tool s_name )
     => t | nil
```

Description

Deletes a simulator option.

Arguments

o_tool	Simulation tool object.
s_name	Name of the simulator option you want to delete.

Value Returned

`t`	Returns `t` when the option is deleted.
`nil`	Returns an error message and `nil` if the option does not exist.

Example

```
asiDeleteSimOption( tool 'delmax )
```

Removes the *delmax* simulator option.

Related Function

To display the current set of simulator options, see the asiDisplaySimOption function on page 8-284.

asiDisplaySimOption

```
asiDisplaySimOption( { o_tool | o_session } )
     => t | nil
```

Description

Displays the current set of simulator option names and values. Use this function only to determine which simulator options you want to modify. Do not use this function as part of another procedure.

Arguments

o_tool | *o_session*	Simulation tool object.

Value Returned

`t`	Displays the set of simulator option names and values and returns `t`.
`nil`	Returns `nil` if there is an error.

Example

```
asiDisplaySimOption( asiGetTool( 'spectreS ) )
```

Displays the *spectreS* simulator options.

asiDisplaySimOptionFormProperties

```
asiDisplaySimOptionFormProperties( o_tool )
     => t | nil
```

Description

Displays the characteristics of the Simulator Options form. Use this function only to determine which form characteristics you want to modify. Do not use this function as part of another procedure.

Arguments

o_tool	Simulation tool object.

Value Returned

`t`	Displays a list of Simulator Option form characteristics and returns `t`.
`nil`	Returns `nil` otherwise.

Example

```
asiDisplaySimOptionFormProperties( asiGetTool('spectreS) )
```

Displays the form characteristics of the Spectre Simulator Options form and returns `t`.

asiGetReservedWordList

```
asiGetReservedWordList( o_session )
```

Description

Returns the simulator specific reserved keyword list. All the keywords specified by `asiGetReservedWordList` will be taken as keywords and not design variables. This process will ensure that the design parameters are not printed in the simulation control file under ‘parameters’ statement. By default, this function returns `nil`.

Arguments

o_session	Simulation tool object.

Value Returned

`l_list`	Returns the simulator specific keyword list.

Note: While adding a simulator in Analog Design environment, you need to overload the function `asiGetReservedWordList` . Otherwise it will return the default value.

asiIsCaseSensitive

```
asiIsCaseSensitive( o_session )
=> t | nil
```

Description

Determines whether the simulator is case sensitive or not. Returns `t`, if the simulator is case sensitive, else returns `nil`. The default value for this function is `t`.

Arguments

o_session	Simulation tool object.

Value Returned

`t`	Determines whether the simulator is case sensitive or not and returns `t`.
`nil`	Returns `nil` otherwise.

Note: While adding a simulator in Analog Design environment, you need to overload the function `asiIsCaseSensitive`. Otherwise it will return the default value.

asiGetSimOptionChoices

```
asiGetSimOptionChoices(
     {o_session | o_tool}
     s_name
     )
     => l_choices | nil
```

Description

Gets the list of choices for a simulator option that is set up as a list box.

Arguments

o_session	Simulation session object.
o_tool	Simulation tool object.
s_name	Name of the option for which you want the list of choices.

Value Returned

l_choices	Returns the list of values for the specified simulation option.
nil	Returns nil if the option does not exist.

Related Function

To display the current set of simulator options, see the asiDisplaySimOption function on page 8-284.

asiGetSimOptionNameList

```
asiGetSimOptionNameList( o_tool )
     => l_nameList
```

Description

Returns the list of simulator option names.

Arguments

o_tool	Simulation tool object.

Value Returned

l_nameList	Returns a list of the names of the simulator options.

Example

Loops through the simulator options and gets the corresponding values and sendMethods for those options.

```
foreach( name asiGetSimOptionNameList(session)
    value = asiGetSimOptionVal( session name)
    sendMethod = asiGetSimOptionSendMethod( session name)
    ; Refer to asiSendOptions for a more complete example.
)
```

asiGetSimOptionSendMethod

```
asiGetSimOptionSendMethod(
     {o_session | o_tool}
     s_name
     )
     => s_sendMethod | nil
```

Description

Gets the *sendMethod* for the specified simulator option. The *sendMethod* indicates how the simulator option is sent to Cadence SPICE.

Arguments

o_session	Simulation session object.
o_tool	Simulation tool object.
s_name	Name of the simulator option.

Value Returned

s_sendMethod	Returns the *sendMethod*, which indicates how the simulator option is sent to Cadence SPICE.
nil	Returns nil if the option does not exist.

Example

```
asiGetSimOptionSendMethod( session 'abstol )
```

Gets the *sendMethod* for the *abstol* simulator option.

asiGetSimOptionVal

```
asiGetSimOptionVal(
     {o_session | o_tool}
     s_name
     )
     => g_value | nil
```

Description

Gets the value for the specified simulator option.

Arguments

o_session	Simulation session object.
o_tool	Simulation tool object.
s_name	Name of the simulator option.

Value Returned

g_value	Returns the value for the simulator option you specify.
nil	Returns nil if the option does not exist.

Example

```
asiGetSimOptionVal( session 'reltol )
```

Returns the value for the *reltol* simulator option.

asiGetSimulationRunCommand

```
asiGetSimulationRunCommand( o_session )
     => g_command | nil
```

Description

Gets the simulation run command. For direct simulators, it also creates the `runSimulation` file.

Arguments

o_session	Simulation session object.

Value Returned

g_command	Returns the simulation run command.
`nil`	Returns `nil` if an error occurs.

Example

```
asiGetSimulationRunCommand( sess )
```

For the spectre session `sess`, this command creates the runSimulation file in the netlist directory and returns `./runSimulation` as the simulation run command.

asiInit<yourSimulator>SimOption

```
asiInit<yourSimulator>SimOption( o_tool )
     => t
```

Description

Calls the procedures to add your simulator options.

Note: You must write `asiInit<yourSimulator>SimOption`, where <*yourSimulator*> is the name of your simulator. Do not include the angle brackets (<>).

Arguments

o_tool	Simulation tool object.

Value Returned

`t`	Returns `t` when your procedures are called.

Note: You must write this procedure to return `t`.

Example

```
procedure( asiInitXYZSimOption( tool )
    <insert your code>
    t
)
```

Creates the procedure that calls the procedures to add the simulator options for the XYZ simulator.

asiSetHostOptions

```
asiSetHostOptions(
     o_session
     t_hostMode
     [t_host]
     [t_remoteDir]
     )
     => t/nil
```

Description

Changes the host mode, host and remote directory for simulation.

Arguments

o_session	Simulation session object.
o_hostMode	The type of simulation you want to select. Valid Values: local, remote and distributed. Default Value: local
t_host	Name of the host on which you want to run the digital simulator.
t_remoteDir	Specifies the project directory on the remote host to be used for remote simulation.

Value Returned

t	Returns `t` when sucessfully changes the Host options.
nil	Returns `nil` otherwise.

Examples

```
asiSetHostOptions( stdobj@0x18d7caa4 "local")
=> t

asiSetHostOptions( stdobj@0x18d7caa4 "distributed")
=> t

asiSetHostOptions(stdobj@0x18d7caa4 "remote" "ciclinux71" "/servers/scratch02/
aakhil/testcase/simulation")
=> t
```

```
asiSetHostOptions(stdobj@0x18d7caa4 "remote")
WARNING (ADE-3042): Unable to contact host (machine)
=> nil
```

asiSetSimOptionChoices

```
asiSetSimOptionChoices(
     {o_session | o_tool}
     s_name
     l_choices
     )
     => l_choices | nil
```

Description

Specifies the list of choices to appear in the list box field for the specified simulator option.

Arguments

o_session	Simulation session object.
o_tool	Simulation tool object.
s_name	Name a simulator option of the type `listBox`.
l_choices	List of choices to appear in the list box field.

Value Returned

l_choices	Returns the new list of choices to appear in the list box field.
`nil`	Returns `nil` if the option does not exist.

asiSetSimOptionVal

```
asiSetSimOptionVal(
     {o_session | o_tool}
     s_name
     g_value
     )
     => g_value | nil
```

Description

Sets the value of the specified simulator option.

Arguments

o_session	Simulation session object.
o_tool	Simulation tool object.
s_name	Name of the simulator option.
g_value	Value for the simulator option.

Value Returned

g_value	Returns the new value for the simulator option.
nil	Returns an error message and nil if the option does not exist.

Example

```
asiSetSimOptionVal( session 'tempdc "35" )
```

Sets the *tempdc* simulator option to 35.

asiGetSimulatorSrcList

```
asiGetSimulatorSrcList(
     o_session
     )
     => l_result
```

Description

Customizes the values in the *Function* drop-down list box of the *Setup Analog Stimuli* form.

Arguments

o_session	Simulation session object.

Value Returned

l_result	List of values in the *Function* drop-down list box of the *Setup Analog Stimuli* form.

Example

```
defmethod ( asiGetSimulatorSrcList ( (session xyz_session) )
   '("dc" "pulse" "sin" "exp")
)
```

9

Analysis Functions

This chapter describes the functions that let you declare and manage simulator analyses.

If you want to use the `asiCheck` function to check values in your analyses or your analysis options, refer to Chapter 16, “Miscellaneous Functions. ”

asiAddAnalysis

```
asiAddAnalysis(
     o_tool
     ?name          s_analysisName
     [?prompt       t_prompt]
     [?fieldList    l_analysisFields]
     [?optionList   l_analysisOptions]
     [?formType     s_formType]
     [?enable       s_enable]
     )
     => o_analysis | nil
```

Description

Adds a new analysis.

Arguments

o_tool	Simulation tool object.
s_analysisName	Name of the analysis.
t_prompt	Prompt for the analysis on the form.
l_analysisFields	List of analysis fields.
l_analysisOptions	List of analysis options.
s_formType	Specifies how the form is displayed. Valid Values:

'oneD	Specifies a sequential display of fields in one column.
'twoD	Specifies a two dimensional display of fields based on row and column positions. The row and column positions are specified with `asiAddAnalysisField`, `asiChangeAnalysisField`, or `asiCreateAnalysisField`.

`'custom`	Lets you specify exact coordinate locations for each field. The coordinate locations are specified with `asiAddAnalysisField`, `asiChangeAnalysisField`, or `asiCreateAnalysisField`. Default Value: `'oneD`
`s_enable`	Boolean flag that specifies whether the analysis is enabled by default. Valid Values: `t` specifies that the analysis is enabled by default, `nil` specifies that the analysis is not enabled by default. Default Value: `nil`

Value Returned

`o_analysis`	Returns the new analysis object if successful.
`nil`	Returns `nil` otherwise.

Example

```
asiAddAnalysis( tool
    ?name 'XYZ
    ?prompt "XYZ"
    ?fieldList list(
        asiCreateAnalysisField(
            ?name 'from
            ?prompt "from"
            ?value "0"
            ?row 1
            ?column 1
        )
        asiCreateAnalysisField(
            ?name 'to
            ?prompt "to"
            ?row 1
            ?column 2
        )
        asiCreateAnalysisField(
            ?name 'by
            ?prompt "by"
            ?row 1
            ?column 3
        )
    )
    ?optionList list(
        asiCreateAnalysisOption(
            ?name 'XYZ1
            ?value "1e-4"
```

```
        )
        asiCreateAnalysisOption(
            ?name 'XYZ2
            ?value "1e-6"
        )
    )
)
```

Adds a new analysis called *XYZanalysis*, which has a *from*, *to,* and *by* field, and *XYZ1* and *XYZ2* analysis options.

Related Function

To display the available analyses, see the asiDisplayAnalysis function on page 9-345.

asiAddAnalysisField

```
asiAddAnalysisField(
     o_analysis
     ?name          s_fieldName
     [?prompt       t_prompt]
     [?type         s_type]
     [?choices      l_choices]
     [?itemsPerRow  x_itemsPerRow]
     [?value        g_value]
     [?min          g_min]
     [?max          g_max]
     [?allowExpr    s_allowExpr]
     [?row          x_row]
     [?column       x_column]
     [?width        x_width]
     [?coordinates  l_coordinates]
     [?displayOrder x_displayOrder]
     [?labelText    t_labelText]
     [?private      s_private]
     [?display      s_display]
     [?editable     s_editable]
     [?appCB        s_appCB]
     [?callback     t_callback]
     [?formApplyCB  s_formApplyCB]
     [?changeCB     st_changeCB]
     [?doubleClickCB  st_doubleClickCB]
     [?numRows      x_numRows]
     [?multipleSelect s_multipleSelect]
     [?invalidateFunc s_invalidateFunc]
     )
     => o_envVar | nil
```

Description

Adds an analysis field to an existing analysis.

Arguments

o_analysis	Analysis object to which you want to add a new field.
s_fieldName	Name of the analysis field to add.
t_prompt	Optional argument that specifies the prompt (on the UI form) for the given option. Default Value: *s_name*

s_type
Type of the option.
Valid Values: `string`, `integer`, `float`, `toggle`, `cyclic`, `radio`, `boolean`, `list`, `radioToggle`, `listBox`, `fileName` (string type for file names only), label (for the label on the UI form), `frame` (for a graphic frame around a field), `separator` (for the separator line on the UI form), `button` (for a button on the UI form), `scale` (for a slider field on the UI form)
Default Value: `string`

Note: See the asiAddSimOption function in the "Simulator Option Functions" chapter for an example that shows how to use the *separator* argument.

l_choices
List of choices if *s_type* is `cyclic`, `radio`, `radioToggle` or `listBox`, or the list of switches if *s_type* is `toggle`. This argument is valid only if *s_type* is `cyclic`, `toggle`, `radio`, `radioToggle`, or `listBox`.

x_itemsPerRow
Numbers of choices per row for `radio`, `cyclic`, `toggle`, and `radioToggle` fields.
Default Value: Total number of choices specified in *l_choices*

g_value
Default value of the option.

g_min
Specifies the minimum value of an `integer`, `float`, or `scale` option.
Default Value: `nil`, which means `-infinity`

g_max
Specifies the maximum value of an `integer`, `float`, or `scale` option.
Default Value: `nil`, which means `+infinity`

s_allowExpr
Specifies whether *g_value* can contain expressions.
Valid Values: `t` (value can contain expressions), `nil` (value cannot contain expressions)
Default Value: `nil`

x_row
Row in the form where the field appears. This argument is valid only for *'twoD* type forms.

x_column
Column in the form where the field appears. The fields are created according to their required widths and are not meant to align with fields in other rows of the form. This argument is valid only for *'twoD* type forms.

x_width

Specifies the width of the field in relation to other fields on the form. Numbers that you enter for this argument are relative values whose actual values are determined by the amount of space available. For example, assume there are three fields for a row that is 400 pixels wide. If the first two fields have an *x_width* of 1, and the last field has an *x_width* of 2, then the widths for the fields are as follows: 100, 100, and 200 pixels. This argument is valid only for *'twoD* type forms.
Default Value: 1

l_coordinates

List specifying the coordinates for the field on the UI form. (The format is the same as for the corresponding *hi* field.) This argument is valid only for *'custom* type forms.

x_displayOrder

Position (from the top) of the option in the form. Use *x_displayOrder* to reposition your options in an *inherited* form. By default, the options appear in the order defined; therefore, inherited options appear first.

The value of *x_displayOrder* must be an integer that specifies the position you want relative to the top of the form. If more than one option has the same display order integer, the last option found (yours) takes precedence. The remaining options shift down on the form. This argument is valid only for *'oneD* type forms.
Valid Values: Any integer

t_labelText

Optional label displayed with frame type fields.
Default Value: `nil`

s_private

Optional argument that declares an option as private, which means the option is not visible to the user. Private options are not part of the UI and their values are not saved when a user saves the environment. You might use this argument for values that are constant for all users of the software. Valid Values: `t` (option does not appear in the UI), `nil` (option appears in the UI)
Default Value: `nil`

s_display

Specifies an expression that determines whether the field is to be displayed on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on

any field in the form.
Default Value: `t`

s_editable Specifies an expression that determines whether the field is to be editable on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

Note: This argument only applies to type-in fields.

s_appCB Specifies a callback function that is executed when the value of the option is changed.
Callback parameter list: (*o_session*)

t_callback Specifies a callback function that is executed when a field is changed on the form while the form is displayed. (You can use this function to do error checking on the value entered by the designer.)

s_formApplyCB Specifies a callback function that is executed when the designer clicks on *Apply* or *OK* on the form containing the associated field.
Callback parameter list: (*o_session r_form r_field*)

st_changeCB Specifies a callback function that is executed when the value of a `listBox` type field is changed on the form.

st_doubleClickCB
Specifies a callback function that is executed when a designer double clicks on a `listBox` type field.

x_numRows Number of rows shown on the form for a `listBox` type field.

s_multipleSelect
Boolean flag that specifies whether multiple items can be selected from the `listBox` type field.
Valid Values: `t` (multiple items can be selected), `nil` (only one item can be selected at a time)

s_invalidateFunc
Specifies a function that is executed when the value of the option is changed. This function invalidates a step in the flowchart.
Callback parameter list: (*o_session*)

Value Returned

o_envVar	Returns the environment variable object representing the field.
nil	Returns nil otherwise.

Example

```
o_analysisAC = asiGetAnalysis(tool 'ac)
asiAddAnalysisField( o_analysisAC
    ?name 'sweep
    ?prompt "Sweep"
    ?type 'cyclic
    ?choices '("Temperature" "Device Parameter"
                  "Model Parameter")
    ?value "Model Parameter"
)
```

Adds a sweep field to an AC analysis.

Related Function

To display the current analysis field names, see the asiDisplayAnalysisField function on page 9-346.

asiAddAnalysisOption

```
asiAddAnalysisOption(
     o_analysis
     ?name          s_optionName
     [?prompt       t_prompt]
     [?type         s_type]
     [?choices      l_choices]
     [?itemsPerRow  x_itemsPerRow]
     [?value        g_value]
     [?min          g_min]
     [?max          g_max]
     [?allowExpr    s_allowExpr]
     [?row          x_row]
     [?column       x_column]
     [?width        x_width]
     [?coordinates  l_coordinates]
     [?displayOrder x_displayOrder]
     [?labelText    t_labelText]
     [?private      s_private]
     [?display      s_display]
     [?editable     s_editable]
     [?appCB        s_appCB]
     [?callback     t_callback]
     [?formApplyCB  s_formApplyCB]
     [?changeCB     st_changeCB]
     [?doubleClickCB   st_doubleClickCB]
     [?numRows      x_numRows]
     [?multipleSelect  s_multipleSelect]
     [?browse       g_browse]
     [?mode         t_browseMode]
     [?invalidateFunc  s_invalidateFunc]
     [?sendMethod      s_sendMethod]
     )
     => o_envVar | nil
```

Description

Adds an option to an existing analysis.

Arguments

o_analysis	Analysis object to which you want to add an option.
s_optionName	Name of the analysis option to add.

t_prompt	Optional argument that specifies the prompt (on the UI form) for the given option. Default Value: *s_name*
s_type	Type of the option. Valid Values: `string, integer, float, toggle, cyclic, radio, boolean, list, radioToggle, listBox, fileName` (string type for file names only), `label` (for the label on the UI form), `frame` (for a graphic frame around a field), separator (for the separator line on the UI form), `button` (for a button on the UI form), `scale` (for a slider field on the UI form) Default Value: `string`

Note: See the asiAddSimOption function in the "Simulator Option Functions" chapter for an example that shows how to use the *separator* argument.

l_choices	List of choices if *s_type* is `cyclic, radio, radioToggle` or `listBox`, or the list of switches if *s_type* is `toggle`. This argument is valid only if *s_type* is `cyclic, toggle, radio, radioToggle,` or `listBox`.
x_itemsPerRow	Numbers of choices per row for `radio, cyclic, toggle,` and `radioToggle` fields. Default Value: Total number of choices specified in *l_choices*
g_value	Default value of the option.
g_min	Specifies the minimum value of an `integer, float`, or `scale` option. Default Value: `nil`, which means `-infinity`
g_max	Specifies the maximum value of an `integer, float,` or `scale` option. Default Value: `nil`, which means `+infinity`
s_allowExpr	Specifies whether *g_value* can contain expressions. Valid Values: `t` (value can contain expressions), `nil` (value cannot contain expressions) Default Value: `nil`
x_row	Row in the form where the field appears. This argument is valid only for *'twoD* type forms.

x_column	Column in the form where the field appears. The fields are created according to their required widths and are not meant to align with fields in other rows of the form. This argument is valid only for *'twoD* type forms.
x_width	Specifies the width of the field in relation to other fields on the form. Numbers that you enter for this argument are relative values whose actual values are determined by the amount of space available. For example, assume there are three fields for a row that is 400 pixels wide. If the first two fields have an *x_width* of 1, and the last field has an *x_width* of 2, then the widths for the fields are as follows: 100, 100, and 200 pixels. This argument is valid only for *'twoD* type forms. Default Value: 1
l_coordinates	List specifying the coordinates for the field on the UI form. (The format is the same as for the corresponding *hi* field.) This argument is valid only for *'custom* type forms.
x_displayOrder	Position (from the top) of the option in the form. Use *x_displayOrder* to reposition your options in an *inherited* form. By default, the options appear in the order defined; therefore, inherited options appear first. The value of *x_displayOrder* must be an integer that specifies the position you want relative to the top of the form. If more than one option has the same display order integer, the last option found (yours) takes precedence. The remaining options shift down on the form. This argument is valid only for *'oneD* type forms. Valid Values: Any integer
t_labelText	Optional label displayed with frame type fields. Default Value: `nil`
s_private	Optional argument that declares an option as private, which means the option is not visible to the user. Private options are not part of the UI and their values are not saved when a user saves the environment. You might use this argument for values that are constant for all users of the software. Valid Values: `t` (option does

not appear in the UI), `nil` (option appears in the UI)
Default Value: `nil`

s_display Specifies an expression that determines whether the field is to be displayed on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

s_editable Specifies an expression that determines whether the field is to be editable on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

Note: This argument only applies to type-in fields.

s_appCB Specifies a callback function that is executed when the value of the option is changed.
Callback parameter list: (*o_session*)

t_callback Specifies a callback function that is executed when a field is changed on the form while the form is displayed. (You can use this function to do error checking on the value entered by the designer.)

s_formApplyCB
Specifies a callback function that is executed when the designer clicks on *Apply* or *OK* on the form containing the associated field.
Callback parameter list: (*o_session r_form r_field*)

st_changeCB Specifies a callback function that is executed when the value of a `listBox` type field is changed on the form.

st_doubleClickCB
Specifies a callback function that is executed when a designer double clicks on a *listBox* type field.

x_numRows Number of rows shown on the form for a `listBox` type field.

s_multipleSelect
Boolean flag that specifies whether multiple items can be selected from the `listBox` type field.

Valid Values: `t` (multiple items can be selected), `nil` (only one item can be selected at a time)

g_browse — Adds a browse button (...) for the specified option. This button can be used to open a file selection form to browse and select a file. The mode of file selection is specified by the *t_mode* argument.

Valid values: `t`, `nil`
Default value: `nil`

t_mode — Specifies mode for the file selection form that is displayed when the *g_browse* argument is set to `t`.

Valid values: `anyFile` specifies that you can open any file type; `existingFile` specifies that you can open any existing file; and `existingFiles` specifies that you can select multiple files.
Default value: `anyFile`

s_invalidateFunc — Specifies a function that is executed when the value of the option is changed. This function invalidates a step in the flowchart.
Callback parameter list: (*o_session*)

s_sendMethod — Specifies how simulator options are sent to Cadence SPICE. (Use this argument only if your simulator is in the SPICE Socket.)
Valid Values:

`set`	Specifies options known by Cadence SPICE (for example, `tempdc` and `dcoppt`).
`ptprop`	Specifies numbers.
`psprop`	Specifies strings.

Default Value: `ptprop`

Note: You can specify other values for the *sendMethod* argument. However, options must be sent to Cadence SPICE with the appropriate `set, ptprop,` or `psprop` statements.

Value Returned

o_envVar	Returns the environment variable object representing the analysis option.
`nil`	Returns `nil` otherwise.

Example

```
asiAddAnalysisOption( o_XYZ
?name 'XYZ2
?value "1e-6"
)
```

Adds the *XYZ2* analysis option to the *XYZ* analysis.

Related Functions

To display the current analysis option names, see the asiDisplayAnalysisOption function on page 9-347.

To change the display characteristics for an analysis options form, see the asiChangeAnalysisOptionFormProperties function on page 9-327.

asiChangeAnalysis

```
asiChangeAnalysis(
     o_tool
     ?name     s_analysisName
     [?prompt  t_prompt]
     [?fieldList    l_analysisFields]
     [?optionList   l_analysisOptions]
     [?formType     s_formType]
     )
     => o_analysis | nil
```

Description

Changes an existing analysis.

Arguments

o_tool	Simulation tool object.
s_analysisName	Name of the analysis you want to change.
t_prompt	Prompt for the analysis on the form.
l_analysisFields	List of analysis fields.
l_analysisOptions	List of analysis options.
s_formType	Specifies how the form is displayed. Valid Values:

'oneD	Specifies a sequential display of fields in one column.
'twoD	Specifies a two dimensional display of fields based on row and column positions. The row and column positions are specified with `asiAddAnalysisField`, `asiChangeAnalysisField`, or `asiCreateAnalysisField`.

`'custom`	Lets you specify exact coordinate locations for each field. The coordinate locations are specified with `asiAddAnalysisField`, `asiChangeAnalysisField`, or `asiCreateAnalysisField`.

Default Value: `'oneD`

Value Returned

`o_analysis`	Returns the analysis object.
`nil`	Returns `nil` otherwise.

Example

```
asiChangeAnalysis( tool
          ?name 'ac
          ?prompt "Modified AC Analysis"
          ?fieldList list(

               asiCreateAnalysisField(
                         ?name 'model
                         ?prompt "Model Name"
                         ?row 4
                         ?column 1
               )
               asiCreateAnalysisField(
                         ?name 'modelParam
                         ?prompt "Model Parameter"
                         ?row 4
                         ?column 2
                    )
               )
     )
```

Changes the form prompt for the AC analysis and adds two fields.

Related Function

To display the available analyses, see the asiDisplayAnalysis function on page 9-345.

asiChangeAnalysisField

```
asiChangeAnalysisField(
     o_analysis
     ?name          s_fieldName
     [?prompt       t_prompt]
     [?type         s_type]
     [?choices      l_choices]
     [?itemsPerRow  x_itemsPerRow]
     [?value        g_value]
     [?min          g_min]
     [?max          g_max]
     [?allowExpr    s_allowExpr]
     [?row          x_row]
     [?column       x_column]
     [?width        x_width]
     [?coordinates  l_coordinates]
     [?displayOrder x_displayOrder]
     [?labelText    t_labelText]
     [?private      s_private]
     [?display      s_display]
     [?editable     s_editable]
     [?appCB        s_appCB]
     [?callback     t_callback]
     [?formApplyCB  s_formApplyCB]
     [?changeCB     st_changeCB]
     [?doubleClickCB   st_doubleClickCB]
     [?numRows      x_numRows]
     [?multipleSelect  s_multipleSelect]
     [?invalidateFunc  s_invalidateFunc]
     )
     => o_envVar | nil
```

Description

Changes a field in an existing analysis.

Arguments

o_analysis	Analysis object with the field you want to change.
s_fieldName	Name of the analysis field to change.
t_prompt	Optional argument that specifies the prompt (on the UI form) for the given option. Default Value: *s_name*

s_type
Type of the option.
Valid Values: `string, integer, float, toggle, cyclic, radio, boolean, list, radioToggle, listBox, fileName` (string type for file names only), `label` (for the label on the UI form), `frame` (for a graphic frame around a field), `separator` (for the separator line on the UI form), `button` (for a button on the UI form), `scale` (for a slider field on the UI form)
Default Value: `string`

Note: See the asiAddSimOption function in the "Simulator Option Functions" chapter for an example that shows how to use the *separator* argument.

l_choices
List of choices if *s_type* is cyclic, radio, radioToggle or listBox, or the list of switches if *s_type* is toggle. This argument is valid only if *s_type* is cyclic, toggle, radio, radioToggle, or listBox.

x_itemsPerRow
Numbers of choices per row for `radio, cyclic, toggle,` and `radioToggle` fields.
Default Value: Total number of choices specified in *l_choices*

g_value
Default value of the option.

g_min
Specifies the minimum value of an `integer, float,` or `scale` option.
Default Value: `nil`, which means `-infinity`

g_max
Specifies the maximum value of an `integer`, `float`, or `scale` option.
Default Value: `nil`, which means `+infinity`

s_allowExpr
Specifies whether *g_value* can contain expressions.
Valid Values: `t` (value can contain expressions), `nil` (value cannot contain expressions)
Default Value: `nil`

x_row
Row in the form where the field appears. This argument is valid only for *'twoD* type forms.

x_column
Column in the form where the field appears. The fields are created according to their required widths and are not meant to align with fields in other rows of the form. This argument is valid only for *'twoD* type forms.

x_width

Specifies the width of the field in relation to other fields on the form. Numbers that you enter for this argument are relative values whose actual values are determined by the amount of space available. For example, assume there are three fields for a row that is 400 pixels wide. If the first two fields have an *x_width* of 1, and the last field has an *x_width* of 2, then the widths for the fields are as follows: 100, 100, and 200 pixels. This argument is valid only for *'twoD* type forms.
Default Value: 1

l_coordinates

List specifying the coordinates for the field on the UI form. (The format is the same as for the corresponding *hi* field.) This argument is valid only for *'custom* type forms.

x_displayOrder

Position (from the top) of the option in the form. Use *x_displayOrder* to reposition your options in an *inherited* form. By default, the options appear in the order defined; therefore, inherited options appear first.

The value of *x_displayOrder* must be an integer that specifies the position you want relative to the top of the form. If more than one option has the same display order integer, the last option found (yours) takes precedence. The remaining options shift down on the form. This argument is valid only for *'oneD* type forms.
Valid Values: Any integer

t_labelText

Optional label displayed with frame type fields.
Default Value: `nil`

s_private

Optional argument that declares an option as private, which means the option is not visible to the user. Private options are not part of the UI and their values are not saved when a user saves the environment. You might use this argument for values that are constant for all users of the software. Valid Values: `t` (option does not appear in the UI), `nil` (option appears in the UI)
Default Value: `nil`

s_display

Specifies an expression that determines whether the field is to be displayed on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on

any field in the form.
Default Value: `t`

s_editable Specifies an expression that determines whether the field is to be editable on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

Note: This argument only applies to type-in fields.

s_appCB Specifies a callback function that is executed when the value of the option is changed.
Callback parameter list: (*o_session*)

t_callback Specifies a callback function that is executed when a field is changed on the form while the form is displayed. (You can use this function to do error checking on the value entered by the designer.)

s_formApplyCB
Specifies a callback function that is executed when the designer clicks on *Apply* or *OK* on the form containing the associated field.
Callback parameter list: (*o_session r_form r_field*)

st_changeCB Specifies a callback function that is executed when the value of a `listBox` type field is changed on the form.

st_doubleClickCB
Specifies a callback function that is executed when a designer double clicks on a `listBox` type field.

x_numRows Number of rows shown on the form for a `listBox` type field.

*s_multipleSelec*t
Boolean flag that specifies whether multiple items can be selected from the `listBox` type field.
Valid Values: `t` (multiple items can be selected), `nil` (only one item can be selected at a time)

s_invalidateFunc
Specifies a function that is executed when the value of the option

is changed. This function invalidates a step in the flowchart. Callback parameter list: (*o_session*)

Value Returned

o_envVar	Returns the new environment variable object representing the analysis field.
`nil`	Returns `nil` otherwise.

Example

```
analysis = asiGetAnalysis(tool 'tran)
asiChangeAnalysisField(analysis
    ?name       'from
    ?prompt     "start"
)
```

Changes the `from` *prompt* to *start*.

Related Function

To display the current analysis field names, see the asiDisplayAnalysisField function on page 9-346.

asiChangeAnalysisOption

```
asiChangeAnalysisOption(
     o_analysis
     ?name          s_optionName
     [?prompt       t_prompt]
     [?type         s_type]
     [?choices      l_choices]
     [?itemsPerRow  x_itemsPerRow]
     [?value        g_value]
     [?min          g_min]
     [?max          g_max]
     [?allowExpr    s_allowExpr]
     [?row          x_row]
     [?column       x_column]
     [?width        x_width]
     [?coordinates  l_coordinates]
     [?displayOrder x_displayOrder]
     [?labelText    t_labelText]
     [?private      s_private]
     [?display      s_display]
     [?editable     s_editable]
     [?appCB        s_appCB]
     [?callback     t_callback]
     [?formApplyCB  s_formApplyCB]
     [?changeCB     st_changeCB]
     [?doubleClickCB     st_doubleClickCB]
     [?numRows      x_numRows]
     [?multipleSelect    s_multipleSelect]
     [?invalidateFunc    s_invalidateFunc]
     [?sendMethod   s_sendMethod]
     )
     => o_envVar | nil
```

Description

Changes an analysis option for an existing analysis.

Arguments

o_analysis	Analysis object with the option you want to change.
s_optionName	Name of the analysis option to change.
t_prompt	Optional argument that specifies the prompt (on the UI form) for the given option. Default Value: *s_name*

s_type
Type of the option.
Valid Values: `string, integer, float, toggle, cyclic, radio, boolean, list, radioToggle, listBox, fileName` (string type for file names only), `label` (for the label on the UI form), `frame` (for a graphic frame around a field), `separator` (for the separator line on the UI form), `button` (for a button on the UI form), `scale` (for a slider field on the UI form)
Default Value: `string`

Note: See the asiAddSimOption function in the "Simulator Option Functions" chapter for an example that shows how to use the *separator* argument.

l_choices
List of choices if *s_type* is `cyclic, radio, radioToggle` or `listBox`, or the list of switches if *s_type* is `toggle`. This argument is valid only if *s_type* is `cyclic, toggle, radio, radioToggle`, or `listBox`.

x_itemsPerRow
Numbers of choices per row for `radio, cyclic, toggle`, and `radioToggle` fields.
Default Value: Total number of choices specified in *l_choices*

g_value
Default value of the option.

g_min
Specifies the minimum value of an `integer, float`, or `scale` option.
Default Value: `nil`, which means `-infinity`

g_max
Specifies the maximum value of an `integer`, `float`, or `scale` option.
Default Value: `nil`, which means `+infinity`

s_allowExpr
Specifies whether *g_value* can contain expressions.
Valid Values: `t` (value can contain expressions), `nil` (value cannot contain expressions)
Default Value: `nil`

x_row
Row in the form where the field appears. This argument is valid only for *'twoD* type forms.

x_column
Column in the form where the field appears. The fields are created according to their required widths and are not meant to

	align with fields in other rows of the form. This argument is valid only for 'twoD type forms.
x_width	Specifies the width of the field in relation to other fields on the form. Numbers that you enter for this argument are relative values whose actual values are determined by the amount of space available. For example, assume there are three fields for a row that is 400 pixels wide. If the first two fields have an *x_width* of 1, and the last field has an *x_width* of 2, then the widths for the fields are as follows: 100, 100, and 200 pixels. This argument is valid only for 'twoD type forms. Default Value: 1
l_coordinates	List specifying the coordinates for the field on the UI form. (The format is the same as for the corresponding *hi* field.) This argument is valid only for 'custom type forms.
x_displayOrder	Position (from the top) of the option in the form. Use *x_displayOrder* to reposition your options in an *inherited* form. By default, the options appear in the order defined; therefore, inherited options appear first. The value of *x_displayOrder* must be an integer that specifies the position you want relative to the top of the form. If more than one option has the same display order integer, the last option found (yours) takes precedence. The remaining options shift down on the form. This argument is valid only for *'oneD* type forms. Valid Values: Any integer
t_labelText	Optional label displayed with frame type fields. Default Value: `nil`
s_private	Optional argument that declares an option as private, which means the option is not visible to the user. Private options are not part of the UI and their values are not saved when a user saves the environment. You might use this argument for values that are constant for all users of the software. Valid Values: `t` (option does not appear in the UI), `nil` (option appears in the UI) Default Value: `nil`

s_display Specifies an expression that determines whether the field is to be displayed on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

s_editable Specifies an expression that determines whether the field is to be editable on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

Note: This argument only applies to type-in fields.

s_appCB Specifies a callback function that is executed when the value of the option is changed.
Callback parameter list: (*o_session*)

t_callback Specifies a callback function that is executed when a field is changed on the form while the form is displayed. (You can use this function to do error checking on the value entered by the designer.)

s_formApplyCB Specifies a callback function that is executed when the designer clicks on *Apply* or *OK* on the form containing the associated field.
Callback parameter list: (*o_session r_form r_field*)

st_changeCB Specifies a callback function that is executed when the value of a `listBox` type field is changed on the form.

st_doubleClickCB Specifies a callback function that is executed when a designer double clicks on a `listBox` type field.

x_numRows Number of rows shown on the form for a `listBox` type field.

s_multipleSelect Boolean flag that specifies whether multiple items can be selected from the `listBox` type field.
Valid Values: `t` (multiple items can be selected), `nil` (only one item can be selected at a time)

s_invalidateFunc
Specifies a function that is executed when the value of the option is changed. This function invalidates a step in the flowchart. Callback parameter list: (*o_session*)

s_sendMethod Specifies how simulator options are sent to Cadence SPICE. (Use this argument only if your simulator is in the SPICE Socket.) Valid Values:

`set`	Specifies options known by Cadence SPICE (for example, `tempdc` and `dcoppt`).
`ptprop`	Specifies numbers.
`psprop`	Specifies strings.

Default Value: `ptprop`

Note: You can specify other values for the *sendMethod* argument. However, your options will be automatically translated into the appropriate `set`, `ptprop`, or `psprop` statements to be sent to Cadence SPICE.

Value Returned

o_envVar	Returns the new environment variable object that represents the analysis option.
`nil`	Returns `nil` otherwise.

Example

```
asiChangeAnalysisOption( analysis
    ?name       'lvltim
    ?choices '( "0" "1" "2" )
)
```

Modifies the *lvltim* transient option by adding an extra choice ("0") to the *choices* list.

Related Functions

To display the current analysis option names, see the asiDisplayAnalysisOption function on page 9-347.

To change the display characteristics for an analysis options form, see the asiChangeAnalysisOptionFormProperties function on page 9-327.

asiChangeAnalysisOptionFormProperties

```
asiChangeAnalysisOptionFormProperties(
     o_analysis
     [?type    s_type]
     [?width   x_width]
     [?columns x_columns]
     )
     => o_formObj | nil
```

Description

Changes the display characteristics for one of the analysis options forms.

Arguments

o_analysis	Analysis object.
s_type	Specifies the form type. Valid Values:

'oneD	Specifies a sequential display of fields in one column.
'twoD	Specifies a two dimensional display of fields based on row and column positions. (Use the asiAddAnalysisField, asiChangeAnalysisField, or asiCreateAnalysisField functions to specify values for the rows and columns.)
'custom	Lets you specify the exact coordinate locations for each field. (Use the asiAddAnalysisField, asiChangeAnalysisField, or asiCreateAnalysisField functions to specify the coordinates.)
'matrix	Specifies a matrix of equally sized fields.

Default Value: 'oneD

x_width	Width of the form, in pixels. Default Value: The minimum default width of the form is 400 pixels. If the fields require more space than this, the form defaults to the smallest width that can accommodate the fields.

x_columns	Number of columns. Use this argument only with matrix type forms. Default Value: 2

Value Returned

o_formObj	Returns the analysis option form object if successful.
nil	Returns nil otherwise.

Example

```
asiChangeAnalysisOptionFormProperties( asiGetAnalysis( asiGetTool( 'spectreS )
'tran ) ?width 500)
```

For the Spectre simulator, changes the width of the Transient Options form to 500 pixels.

asiCreateAnalysisField

```
asiCreateAnalysisField(
     ?name          s_fieldName
     [?prompt       t_prompt]
     [?type         s_type]
     [?choices      l_choices]
     [?itemsPerRow  x_itemsPerRow]
     [?value        g_value]
     [?min          g_min]
     [?max          g_max]
     [?allowExpr    s_allowExpr]
     [?row          x_row]
     [?column       x_column]
     [?width        x_width]
     [?coordinates  l_coordinates]
     [?displayOrder x_displayOrder]
     [?labelText    t_labelText]
     [?private      s_private]
     [?display      s_display]
     [?editable     s_editable]
     [?appCB        s_appCB]
     [?callback     t_callback]
     [?formApplyCB  s_formApplyCB]
     [?changeCB     st_changeCB]
     [?doubleClickCB     st_doubleClickCB]
     [?numRows      x_numRows]
     [?multipleSelect    s_multipleSelect]
     [?invalidateFunc    s_invalidateFunc]
     )
     => o_envVar | nil
```

Description

Creates a new analysis field, such as *from* or *to*, for a new or changed analysis.

You can call this function from within `asiAddAnalysis` or `asiChangeAnalysis` as an argument to *?fieldList.*

Arguments

s_fieldName	Name of the analysis field to define.
t_prompt	Optional argument that specifies the prompt (on the UI form) for the given option. Default Value: *s_name*

s_type
Type of the option.
Valid Values: `string, integer, float, toggle, cyclic, radio, boolean, list, radioToggle, listBox, fileName` (string type for file names only), `label` (for the label on the UI form), `frame` (for a graphic frame around a field), `separator` (for the separator line on the UI form), `button` (for a button on the UI form), `scale` (for a slider field on the UI form)
Default Value: `string`

Note: See the asiAddSimOption function in the "Simulator Option Functions" chapter for an example that shows how to use the *separator* argument.

l_choices
List of choices if *s_type* is `cyclic, radio, radioToggle` or `listBox`, or the list of switches if *s_type* is `toggle`. This argument is valid only if *s_type* is `cyclic, toggle, radio, radioToggle,` or `listBox`.

x_itemsPerRow
Numbers of choices per row for `radio, cyclic, toggle`, and `radioToggle` fields.
Default Value: Total number of choices specified in *l_choices*

g_value
Default value of the option.

g_min
Specifies the minimum value of an `integer`, `float`, or `scale` option.
Default Value: `nil`, which means `-infinity`

g_max
Specifies the maximum value of an `integer`, `float`, or `scale` option.
Default Value: `nil`, which means `+infinity`

s_allowExpr
Specifies whether *g_value* can contain expressions.
Valid Values: `t` (value can contain expressions), `nil` (value cannot contain expressions)
Default Value: `nil`

x_row
Row in the form where the field appears. This argument is valid only for *'twoD* type forms.

x_column
Column in the form where the field appears. The fields are created according to their required widths and are not meant to

	align with fields in other rows of the form. This argument is valid only for *'twoD* type forms.
x_width	Specifies the width of the field in relation to other fields on the form. Numbers that you enter for this argument are relative values whose actual values are determined by the amount of space available. For example, assume there are three fields for a row that is 400 pixels wide. If the first two fields have an *x_width* of 1, and the last field has an *x_width* of 2, then the widths for the fields are as follows: 100, 100, and 200 pixels. This argument is valid only for *'twoD* type forms. Default Value: 1
l_coordinates	List specifying the coordinates for the field on the UI form. (The format is the same as for the corresponding *hi* field.) This argument is valid only for *'custom* type forms.
x_displayOrder	Position (from the top) of the option in the form. Use *x_displayOrder* to reposition your options in an `inherited` form. By default, the options appear in the order defined; therefore, inherited options appear first. The value of *x_displayOrder* must be an integer that specifies the position you want relative to the top of the form. If more than one option has the same display order integer, the last option found (yours) takes precedence. The remaining options shift down on the form. This argument is valid only for *'oneD* type forms. Valid Values: Any integer
t_labelText	Optional label displayed with frame type fields. Default Value: `nil`
s_private	Optional argument that declares an option as private, which means the option is not visible to the user. Private options are not part of the UI and their values are not saved when a user saves the environment. You might use this argument for values that are constant for all users of the software. Valid Values: `t` (option does not appear in the UI), `nil` (option appears in the UI) Default Value: `nil`

s_display Specifies an expression that determines whether the field is to be displayed on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

s_editable Specifies an expression that determines whether the field is to be editable on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

Note: This argument only applies to type-in fields.

s_appCB Specifies a callback function that is executed when the value of the option is changed.
Callback parameter list: (*o_session*)

t_callback Specifies a callback function that is executed when a field is changed on the form while the form is displayed. (You can use this function to do error checking on the value entered by the designer.)

s_formApplyCB Specifies a callback function that is executed when the designer clicks on *Apply* or *OK* on the form containing the associated field.
Callback parameter list: (`o_session r_form r_field`)

st_changeCB Specifies a callback function that is executed when the value of a `listBox` type field is changed on the form.

st_doubleClickCB Specifies a callback function that is executed when a designer double clicks on a `listBox` type field.

x_numRows Number of rows shown on the form for a `listBox` type field.

s_multipleSelect Boolean flag that specifies whether multiple items can be selected from the *listBox* type field.
Valid Values: `t` (multiple items can be selected), `nil` (only one item can be selected at a time)

s_invalidateFunc

Specifies a function that is executed when the value of the option is changed. This function invalidates a step in the flowchart. Callback parameter list: (*o_session*)

Value Returned

o_envVar	Returns the environment variable object that represents the field.
nil	Returns nil otherwise.

Example

```
asiAddAnalysis( tool
    ?name 'XYZ
    ?fieldList list(
        asiCreateAnalysisField(
            ?name 'from
            ?prompt "from"
            ?value "0"
            )
        asiCreateAnalysisField(
            ?name 'to
            ?prompt "to"
            )
        asiCreateAnalysisField(
            ?name 'by
            ?prompt "by"
            )
        )
    )
```

Adds a new analysis called *XYZ* analysis. This analysis has a *from*, *to,* and *by* field. The following example shows another way to add a new XYZ analysis with a *from*, *to*, and *by* field.

```
fromField = asiCreateAnalysisField(
                        ?name 'from
                        ?prompt "from"
                        ?value "0"
            )
toField = asiCreateAnalysisField(
                        ?name 'to
                        ?prompt "to"
            )
byField = asiCreateAnalysisField(
                        ?name 'by
                        ?prompt "by"
            )
asiAddAnalysis( tool
    ?name 'XYZ
    ?prompt "XYZ"
```

```
    ?fieldList list( fromField toField byField )
)
```

Related Function

To display the current analysis field names, see the asiDisplayAnalysisField function on page 9-346.

asiCreateAnalysisOption

```
asiCreateAnalysisOption(
     ?name          s_optionName
     [?prompt       t_prompt]
     [?type         s_type]
     [?choices      l_choices]
     [?itemsPerRow  x_itemsPerRow]
     [?value        g_value]
     [?min          g_min]
     [?max          g_max]
     [?allowExpr    s_allowExpr]
     [?row          x_row]
     [?column       x_column]
     [?width        x_width]
     [?coordinates  l_coordinates]
     [?displayOrder x_displayOrder]
     [?labelText    t_labelText]
     [?private      s_private]
     [?display      s_display]
     [?editable     s_editable]
     [?appCB        s_appCB]
     [?callback     t_callback]
     [?formApplyCB  s_formApplyCB]
     [?changeCB     st_changeCB]
     [?doubleClickCB     st_doubleClickCB]
     [?numRows      x_numRows]
     [?multipleSelect    s_multipleSelect]
     [?invalidateFunc    s_invalidateFunc]
     [?sendMethod   s_sendMethod]
     )
     => o_envVar | nil
```

Description

Creates a new analysis option for a new or changed analysis. You can call this function from within `asiAddAnalysis` or `asiChangeAnalysis` as an argument to *?optionList*.

Arguments

s_optionName	Name of the analysis option to define.
t_prompt	Optional argument that specifies the prompt (on the UI form) for the given option. Default Value: *s_name*

s_type
Type of the option.
Valid Values: `string, integer, float, toggle, cyclic, radio, boolean, list, radioToggle, listBox, fileName` (string type for file names only), `label` (for the label on the UI form), `frame` (for a graphic frame around a field), `separator` (for the separator line on the UI form), `button` (for a button on the UI form), `scale` (for a slider field on the UI form)
Default Value: `string`

Note: See the asiAddSimOption function in the "Simulator Option Functions" chapter for an example that shows how to use the *separator* argument.

l_choices
List of choices if *s_type* is `cyclic, radio, radioToggle or listBox`, or the list of switches if *s_type* is `toggle`. This argument is valid only if *s_type* is `cyclic, toggle, radio, radioToggle`, or `listBox`.

x_itemsPerRow
Numbers of choices per row for `radio, cyclic, toggle`, and `radioToggle` fields.
Default Value: Total number of choices specified in *l_choices*

g_value
Default value of the option.

g_min
Specifies the minimum value of an `integer, float`, or `scale` option.
Default Value: `nil`, which means `-infinity`

g_max
Specifies the maximum value of an `integer, float`, or `scale` option.
Default Value: `nil`, which means `+infinity`

s_allowExpr
Specifies whether *g_value* can contain expressions.
Valid Values: `t` (value can contain expressions), `nil` (value cannot contain expressions)
Default Value: `nil`

x_row
Row in the form where the field appears. This argument is valid only for *'twoD* type forms.

x_column
Column in the form where the field appears. The fields are created according to their required widths and are not meant to

	align with fields in other rows of the form. This argument is valid only for *'twoD* type forms.
x_width	Specifies the width of the field in relation to other fields on the form. Numbers that you enter for this argument are relative values whose actual values are determined by the amount of space available. For example, assume there are three fields for a row that is 400 pixels wide. If the first two fields have an *x_width* of 1, and the last field has an *x_width* of 2, then the widths for the fields are as follows: 100, 100, and 200 pixels. This argument is valid only for *'twoD* type forms. Default Value: 1
l_coordinates	List specifying the coordinates for the field on the UI form. (The format is the same as for the corresponding *hi* field.) This argument is valid only for *'custom* type forms.
x_displayOrder	Position (from the top) of the option in the form. Use *x_displayOrder* to reposition your options in an *inherited* form. By default, the options appear in the order defined; therefore, inherited options appear first. The value of *x_displayOrder* must be an integer that specifies the position you want relative to the top of the form. If more than one option has the same display order integer, the last option found (yours) takes precedence. The remaining options shift down on the form. This argument is valid only for *'oneD* type forms. Valid Values: Any integer
t_labelText	Optional label displayed with frame type fields. Default Value: `nil`
s_private	Optional argument that declares an option as private, which means the option is not visible to the user. Private options are not part of the UI and their values are not saved when a user saves the environment. You might use this argument for values that are constant for all users of the software. Valid Values: `t` (option does not appear in the UI), `nil` (option appears in the UI) Default Value: `nil`

s_display Specifies an expression that determines whether the field is to be displayed on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

s_editable Specifies an expression that determines whether the field is to be editable on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

Note: This argument only applies to type-in fields.

s_appCB Specifies a callback function that is executed when the value of the option is changed.
Callback parameter list: (*o_session*)

t_callback Specifies a callback function that is executed when a field is changed on the form while the form is displayed. (You can use this function to do error checking on the value entered by the designer.)

s_formApplyCB Specifies a callback function that is executed when the designer clicks on *Apply* or *OK* on the form containing the associated field.
Callback parameter list: (*o_session r_form r_field*)

st_changeCB Specifies a callback function that is executed when the value of a `listBox` type field is changed on the form.

st_doubleClickCB Specifies a callback function that is executed when a designer double clicks on a `listBox` type field.

x_numRows Number of rows shown on the form for a `listBox` type field.

s_multipleSelect Boolean flag that specifies whether multiple items can be selected from the `listBox` type field.
Valid Values: `t` (multiple items can be selected), `nil` (only one item can be selected at a time)

s_invalidateFunc
Specifies a function that is executed when the value of the option is changed. This function invalidates a step in the flowchart. Callback parameter list: (*o_session*)

s_sendMethod Specifies how simulator options are sent to Cadence SPICE. (Use this argument only if your simulator is in the SPICE Socket.) Valid Values:

`set`	Specifies options known by Cadence SPICE (for example, `tempdc` and `dcoppt`).
`ptprop`	Specifies numbers.
`psprop`	Specifies strings.

Default Value: `ptprop`

Note: You can specify other values for the *sendMethod* argument. However, options must be sent to Cadence SPICE with the appropriate `set`, `ptprop`, or `psprop` statements.

Value Returned

o_envVar	Returns the environment variable object representing the analysis option.
`nil`	Returns `nil` otherwise.

Example

Adds a new analysis called *XYZanalysis* with *XYZ1* and *XYZ2* analysis options.

```
asiAddAnalysis( tool
        ?name 'XYZ
        ?prompt "XYZ"
        ?optionList list(
            asiCreateAnalysisOption(
                ?name 'XYZ1
                ?value "1e-4"
            )
            asiCreateAnalysisOption(
                ?name 'XYZ2
                ?value "1e-6"
        )
            )
)
```

The following example shows another way to add new XYZ analysis with *XYZ1* and *XYZ2* analysis options.

```
option1 = asiCreateAnalysisOption(
                    ?name 'XYZ1
                    ?value "1e-4"
            )
option2 = asiCreateAnalysisOption(
                    ?name 'XYZ2
                    ?value "1e-6"
            )
asiAddAnalysis(tool
    ?name 'XYZ
    ?prompt "XYZ"
    ?optionList list( option1 option2 )
)
```

Related Functions

To display the current analysis option names, see the asiDisplayAnalysisOption function on page 9-347.

To change the display characteristics for an analysis options form, see the asiChangeAnalysisOptionFormProperties function on page 9-327.

asiDeleteAnalysis

```
asiDeleteAnalysis( o_tool s_analysisName )
     => t | nil
```

Description

Deletes an analysis.

Arguments

o_tool	Simulation tool object.
s_analysisName	Name of the analysis you want to delete.

Value Returned

t	Returns t when the analysis is deleted.
nil	Returns an error message and nil if the analysis does not exist.

Example

```
tool = asiGetTool('cdsSpice)
asiDeleteAnalysis(tool 'ac)
```

Deletes an AC analysis from the Cadence SPICE tool.

Related Function

To display the current set of analyses, see the asiDisplayAnalysis function on page 9-345.

asiDeleteAnalysisField

```
asiDeleteAnalysisField( o_analysis s_fieldName )
     => t | nil
```

Description

Deletes an analysis field from an existing analysis.

Arguments

o_analysis	Existing analysis object.
s_fieldName	Name of the analysis field to delete.

Value Returned

`t`	Returns `t` when the analysis field is deleted.
`nil`	Returns `nil` if the field does not exist.

Example

```
analysis = asiGetAnalysis(tool 'tran)
asiDeleteAnalysisField(analysis 'by)
```

Deletes the *by* field from the existing transient analysis.

Related Function

To display the current set of analysis field names, see the asiDisplayAnalysisField function on page 9-346.

asiDeleteAnalysisOption

```
asiDeleteAnalysisOption( o_analysis s_optionName )
     => t | nil
```

Description

Deletes an analysis option.

Arguments

o_analysis	Analysis object with the option you want to delete.
s_optionName	Name of the analysis option to delete.

Value Returned

`t`	Returns `t` when the option is deleted.
`nil`	Returns `nil` if the option does not exist.

Example

```
analysis = asiGetAnalysis(tool 'tran)
asiDeleteAnalysisOption(analysis 'tranOpt1)
```

Deletes the *tranOpt1* analysis option from the *ran* analysis.

Related Function

To display the current analysis option names, see the asiDisplayAnalysisOption function on page 9-347.

asiDisableAnalysis

```
asiDisableAnalysis( o_analysis )
     => t | nil
```

Description

Disables an analysis while keeping it in the analysis list. The analysis remains in the UI, but it is not sent to the simulator.

Arguments

o_analysis	Specifies an analysis object.

Value Returned

`t`	Returns `t` when the analysis is disabled.
`nil`	Returns `nil` if the analysis does not exist.

Example

```
analysis = asiGetAnalysis(session 'tran)
asiDisableAnalysis(analysis)
```

Disables the *tran* analysis.

asiDisplayAnalysis

```
asiDisplayAnalysis( o_tool )
     => t | nil
```

Description

Displays the analyses for a tool. Use this function to determine which analyses you need to add or modify. Do not use this function as part of another procedure.

Arguments

o_tool	Simulation tool object.

Value Returned

`t`	Returns `t` and displays the available analyses.
`nil`	Returns `nil` otherwise.

Example

```
tool = asiGetTool('analog)
asiDisplayAnalysis(tool)
```

Displays the analyses registered for the Analog Class.

asiDisplayAnalysisField

```
asiDisplayAnalysisField( o_analysis )
     => t | nil
```

Description

Displays the analysis field names for an analysis. Use this function to determine which analysis field you want to modify. Do not use this function as part of another procedure.

Arguments

o_analysis	Analysis object.

Value Returned

t	Returns t and displays the current analysis field names.
nil	Returns nil otherwise.

Example

```
analysis = asiGetAnalysis(tool 'tran)
asiDisplayAnalysisField(analysis)
```

Displays the names of the analysis fields for the *tran* analysis.

asiDisplayAnalysisOption

```
asiDisplayAnalysisOption( o_analysis )
     => t | nil
```

Description

Displays the analysis option names for an analysis. Use this function to determine which analysis option you want to modify. Do not use this function as part of another procedure.

Arguments

o_analysis	Analysis object.

Value Returned

`t`	Returns `t` and displays the current analysis option names.
`nil`	Returns `nil` otherwise.

Example

```
analysis = asiGetAnalysis(tool 'tran)
asiDisplayAnalysisOption(analysis)
```

Displays the analysis option names for the *tran* analysis.

asiDisplayAnalysisOptionFormProperties

```
asiDisplayAnalysisOptionFormProperties( o_analysis )
     => t | nil
```

Description

Displays the characteristics for one of the analysis options forms. Use this function only to determine which form characteristics you want to modify. Do not use this function as part of another procedure.

Arguments

o_analysis	Analysis object.

Value Returned

`t`	Displays the analysis options form characteristics and returns `t`.
`nil`	Returns `nil` otherwise.

Example

```
asiDisplayAnalysisOptionFormProperties(
asiGetAnalysis( asiGetTool( 'spectreS ) 'tran))
```

Displays the form characteristics for the Transient Options form.

asiEnableAnalysis

```
asiEnableAnalysis( o_analysis )
     => t | nil
```

Description

Enables an analysis, which means the analysis is selected and sent to the simulator.

Arguments

o_analysis	Specifies an analysis object.

Value Returned

`t`	Returns `t` if the analysis instance is enabled.
`nil`	Returns `nil` if the analysis does not exist.

Example

```
analysis = asiGetAnalysis(session 'tran)
asiEnableAnalysis(analysis)
```

Enables the *tran* analysis.

asiFormatAnalysis

```
asiFormatAnalysis( o_analysis p_fp )
     => t | nil
```

Description

Formats the analysis statements to send to Cadence SPICE.

Note for Integrators: This function is defined as a method for the Analog Class. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_analysis	Specifies the analysis object.
p_fp	Specifies a file pointer to the file containing the analysis statements to send to the simulator (for simulators that are not in the Cadence SPICE Socket).

Value Returned

`t`	Returns `t` if the statements are generated.
`nil`	Returns `nil` otherwise.

Example for Integrators

In this example, the integrator of the XYZ simulator sends the trial analysis to Cadence SPICE, which formats the analysis to send to the XYZ simulator.

```
defmethod( asiFormatAnalysis ( (analysis XYZ_trial_analysis) fp )
    let(( str ( aelEnv aelEnvCreate( 's ) ) (session asiGetSession(  analysis ))
)

        str = sprintf( nil "* Trial Analysis\n")
        str = sprintf( nil "%sptprop XYZ_trial DoIt 1\n" str )

        str = sprintf( nil "%sptprop XYZ_trial from %s\n" str
             aelEnvInterpret( aelEnv asiGetAnalysisFieldVal
            (analysis 'from )))
        str = sprintf( nil "%sptprop XYZ_trial to %s\n" str
             aelEnvInterpret( aelEnv asiGetAnalysisFieldVal
            (analysis 'to )))
        str = sprintf( nil "%sptprop XYZ_trial by %s\n" str
             aelEnvInterpret( aelEnv asiGetAnalysisFieldVal
```

```
            (analysis 'by )))
        asiSendSim( session str nil nil nil )
        asiFormatAnalysisOption( analysis fp )
        t
    )
)
```

asiGetAnalysis

```
asiGetAnalysis(
     {o_session | o_tool}
     s_analysisName
     )
     => o_analysis | nil
```

Description

Gets an analysis object.

Arguments

o_session	Simulation session object.
o_tool	Simulation tool object.
s_analysisName	Name of the analysis you want to get.

Value Returned

o_analysis	Returns the analysis object.
nil	Returns nil if the analysis does not exist.

Example

```
analysis = asiGetAnalysis( session 'noise )
```

Returns the *noise* analysis object.

Related Function

To display the current analysis names, see the asiDisplayAnalysis function on page 9-345.

asiGetAnalysisFieldChoices

```
asiGetAnalysisFieldChoices( o_analysis s_fieldName )
     => l_choices | nil
```

Description

Gets the list of choices for an analysis field that is set up as a list box.

Arguments

o_analysis	Specifies an analysis object.
s_fieldName	Name of the analysis field for which you want to get the list of choices.

Value Returned

l_choices	Returns the list of choices for the specified analysis field.
nil	Returns nil if the field does not exist.

Related Function

To display the current set of analysis fields, see the asiDisplayAnalysisField function on page 9-346.

asiGetAnalysisFieldList

```
asiGetAnalysisFieldList( o_analysis )
     => l_fieldList | nil
```

Description

Returns a list of analysis field objects defined for a particular analysis object.

Arguments

o_analysis	Specifies an analysis object.

Value Returned

l_fieldList	Returns the list of field objects for the specified analysis object.
nil	Returns nil if the analysis has no fields associated with it.

Example

```
dcAnal = asiGetAnalysis(session 'dc)
asiGetAnalysisFieldList( dcAnal )
```

Returns the list of DC analysis fields.

Related Function

To display the analysis field names for an analysis, see the asiDisplayAnalysisField function on page 9-346.

asiGetAnalysisFieldVal

```
asiGetAnalysisFieldVal( o_analysis s_fieldName )
     => g_value | nil
```

Description

Gets the value of an analysis field from the environment.

Arguments

o_analysis	Specifies an analysis object.
s_fieldName	Name of the analysis field for which you want to get the value.

Value Returned

g_value	Returns the value for the given field.
nil	Returns nil if the field does not exist.

Example

```
analysis = asiGetAnalysis( session 'tran )
asiGetAnalysisFieldVal( analysis 'from )
```

Gets the value of the from field for an analysis.

Related Function

To display the current analysis field names, see the asiDisplayAnalysisField function on page 9-346.

asiGetAnalysisFormFieldChoices

```
asiGetAnalysisFormFieldChoices( r_form s_analysisName s_fieldName )
     => l_choices | nil
```

Description

Returns the list of choices for a field in the Choosing Analyses form. This procedure can be used within the *asiCheck* method to get the list of choices for an analysis field from the form for subsequent value checking. You can also use this procedure in an expression that controls whether a field is displayed in the form.

Arguments

r_form	Form containing the analysis field.
s_analysisName	Name of the analysis.
s_fieldName	Name of an analysis field of the type `listBox`.

Value Returned

l_choices	Returns the list of choices for the specified field on the Choosing Analyses form.
`nil`	Returns `nil` if the field does not exist.

asiGetAnalysisFormFieldVal

```
asiGetAnalysisFormFieldVal( r_form s_analysisName s_fieldName )
     => g_fieldValue | nil
```

Description

Returns the value of a field in the Choosing Analyses form. This procedure can be used within the *asiCheck* method to get the values of analysis fields from the form for subsequent value checking. You can also use this procedure in an expression that controls whether a field is displayed in the form.

Arguments

r_form	Form containing the analysis field.
s_analysisName	Name of the analysis.
s_fieldName	Name of the analysis field.

Value Returned

g_fieldValue	Returns the value for a field on the Choosing Analyses form.
nil	Returns nil if the field does not exist.

Example

The following example shows how you might use the asiGetAnalysisFormFieldVal function to get the value of the *incrType* field (for an ac analysis) within the display expression

for the *lin* field. In this example, the *lin* field is displayed on the form only when *incrType* is *"Linear".*

```
asiCreateAnalysisField(
    ?name           'lin
    ?prompt         "Step Size (Hz)"
    ?display        'equal(asiGetAnalysisFormFieldVal( form 'ac
                        'incrType ) "Linear")
)
```

The first argument must be explicitly specified for proper evaluation.

See the asiCheck function in the “Miscellaneous Functions” chapter for an example that shows how to use the `asiGetAnalysisFormFieldVal` function within the *asiCheck* method to get the values of analysis form fields for value checking.

asiGetAnalysisName

```
asiGetAnalysisName( o_analysis )
     => s_analysisName | nil
```

Description

Gets the name of the analysis.

Arguments

o_analysis	Specifies an analysis object.

Value Returned

s_analysisName	Returns the name of the analysis.
nil	Returns nil if the analysis is invalid.

asiGetAnalysisNameList

```
asiGetAnalysisNameList({o_session | o_tool}) => l_analysesNames
```

Description

Returns a list of analysis names defined for a tool.

Arguments

o_session	Specifies a simulation session object.
o_tool	Specifies a simulation tool object.

Value Returned

l_analysesNames	Returns a list of analysis names

Example

```
listOfAnalysisNames = asiGetAnalysisNameList(o_session)
```

Returns the list of analysis names for *o_session*.

asiGetAnalysisOptionChoices

```
asiGetAnalysisOptionChoices( o_analysis s_optionName )
     => l_choices | nil
```

Description

Gets the list of choices for an analysis option that is set up as a list box.

Arguments

o_analysis	Specifies an analysis object.
s_optionName	Name of the option for which you want to get the list of choices.

Value Returned

l_choices	Returns the list of choices for the analysis option.
nil	Returns nil if the option does not exist.

Related Function

To display the current analysis option names, see the asiDisplayAnalysisOption function on page 9-347.

asiGetAnalysisOptionList

```
asiGetAnalysisOptionList( o_analysis )
     => l_optionObjects | nil
```

Description

Gets a list of analysis option objects defined for a particular analysis object.

Arguments

o_analysis	Specifies an analysis object.

Value Returned

l_optionObjects	Returns a list of analysis option objects.
nil	Returns nil if the analysis has no options.

Examples

```
asiGetAnalysisOptionList( analysis )
```

Returns the list of analysis options.

The following example shows how you might use the asiGetAnalysisOptionList routine in your asiFormatAnalysisOption routine.

```
defmethod( asiFormatAnalysisOption ( ( analysis XYZ_trial_analysis )
    _fp )

    let(( name value command sendMethod (session asiGetSession
        (analysis )) )

        foreach( option asiGetAnalysisOptionList( analysis )
            name  = asiGetName( option )
            value = asiGetAnalysisOptionVal( analysis name )
            sendMethod = asiGetAnalysisOptionSendMethod( analysis
                         name)

            if( artBlankString( value ) then
                    sprintf( command "deprop XYZ_trial %s\n" name )
            else
                    caseq( sendMethod
                        ( set
```

```
                                sprintf( command "set %s=%s\n" name
                                           value )
                                )
                                ( ptprop
                                    sprintf( command "ptprop XYZ_trial
                                    %s %s\n" name value )
                                )
                        ( psprop
                             sprintf( command "psprop XYZ_trial %s
                                        \"%s\" \n" name value )
                        )
                        ( t
                            warn( "don't know how to send XYZ trial
                                option %s\n." name  )
                            command = nil
                        )
                )
            )

            when( command
                asiSendSim( session command nil nil nil )
            )
         )
        )
)
```

asiGetAnalysisOptionSendMethod

```
asiGetAnalysisOptionSendMethod( o_analysis s_optionName )
     => s_sendMethod
```

Description

Gets the *sendMethod* for an option in an analysis. The *sendMethod* indicates how the analysis option is sent to Cadence SPICE.

Arguments

o_analysis	Specifies an analysis object.
s_optionName	Name of an analysis option for which you want to get the *sendMethod*.

Value Returned

s_sendMethod	Returns the *sendMethod* for the analysis option.

Example

```
asiGetAnalysisOptionSendMethod( analysis 'delmax )
```

Returns the *sendMethod* for the *delmax* analysis option.

asiGetAnalysisOptionVal

```
asiGetAnalysisOptionVal( o_analysis s_optionName )
     => g_value | nil
```

Description

Gets the value for the given option in an analysis.

Arguments

o_analysis	Specifies an analysis object.
s_optionName	Name of the option for which you want to get the value.

Value Returned

g_value	Returns the value for the analysis option.
`nil`	Returns `nil` if the option does not exist.

Example

```
asiGetAnalysisOptionVal( analysis 'delmax )
```

Returns the value for the *delmax* analysis option.

Related Function

To display the current analysis option names, see the asiDisplayAnalysisOption function on page 9-347.

asiGetAnalysisParamNameList

```
asiGetAnalysisParamNameList( o_analysis )
     => l_analysisParamNameList | nil
```

Description

Returns a concatenated list of fields and options defined for an analysis object.

Arguments

o_analysis	Specifies an analysis object.

Value Returned

l_analysisParamNameList	A concatenated list of fields and options defined for the analysis object.
nil	Returns nil if the analysis has no fields or options associated with it.

Example

```
analysis = asiGetAnalysis( session 'tran )
asiGetAnalysisParamNameList(analysis)
```

Displays the names of fields and options defined for tran analysis in a list.

asiGetEnabledAnalysisList

```
asiGetEnabledAnalysisList( o_session ) => l_analysisEnabledList | nil
```

Description

Returns a list of all the enabled analyses.

Arguments

o_session	Specifies a simulation session object.

Value Returned

l_analysisEnabledList	Returns a list of the enabled analyses.
nil	Returns nil if no analysis is enabled.

Example

```
l_analysisEnabledList = asiGetEnabledAnalysisList(o_session)
```

Returns the list of enabled analyses for *o_session*.

asiInit<yourSimulator>Analysis

```
asiInit<yourSimulator>Analysis( o_tool )
     => t
```

Description

Calls the procedures that modify your simulator's analyses.

Note: You must write `asiInit<`*`yourSimulator`*`>Analysis`, where <*`yourSimulator`*> is the name of your simulator. Do not include the angle brackets (<>).

Arguments

`o_tool`	Simulation tool object.

Value Returned

`t`	Returns `t` if successful.

Note: You must write this procedure to return `t`.

Example

```
procedure( asiInitXYZAnalysis( tool )
    <insert your code>
    t
)
```

Creates the procedure that calls the procedures to specify or change the analyses for the *XYZ* simulator.

asiIsAnalysisEnabled

```
asiIsAnalysisEnabled( o_analysis )
     => t | nil
```

Description

Tests to determine whether an analysis is enabled.

Arguments

o_analysis	Specifies an analysis object.

Value Returned

`t`	Returns `t` if the analysis is enabled
`nil`	Returns `nil` if the analysis is not enabled.

Example

```
analysis = asiGetAnalysis(session 'tran)
when( asiIsAnalysisEnabled(analysis)
println("The transient analysis is enabled")
)
```

Prints “The transient analysis is enabled” if `tran` is enabled.

asiSetAnalysisFieldChoices

```
asiSetAnalysisFieldChoices( o_analysis s_fieldName l_choices )
     => l_choices | nil
```

Description

Specifies the list of choices to appear in the list box for an analysis field.

Arguments

o_analysis	Specifies an analysis object.
s_fieldName	Name of the analysis field for which you want to set the value.
l_choices	List of choices to appear in the list box field.

Value Returned

l_choices	Returns the new list of choices for the analysis field.
nil	Returns nil if the field does not exist.

asiSetAnalysisFieldVal

```
asiSetAnalysisFieldVal( o_analysis s_fieldName g_value )
     => g_value | nil
```

Description

Sets the value for a field of an analysis.

Arguments

o_analysis	Specifies an analysis object.
s_fieldName	Name of the field for which you want to set the value.
g_value	Value for the field.

Value Returned

g_value	Returns the new value for the analysis field.
nil	Returns nil if the field does not exist.

Example

```
asiSetAnalysisFieldVal( analysis 'from "5n" )
```

Sets the from field to 5n.

Related Function

To display the current analysis field names, see the asiDisplayAnalysisField function on page 9-346.

asiSetAnalysisFormFieldChoices

```
asiSetAnalysisFormFieldChoices( r_form s_analysisName s_fieldName l_choices )
     => l_choices | nil
```

Description

Sets the list of choices for the specified field on the Choosing Analyses form.

Arguments

r_form	Form containing the analysis fields.
s_analysisName	Name of the analysis.
s_fieldName	Name of an analysis field of the type `listBox`.
l_choices	List of choices for the analysis field.

Value Returned

l_choices	Returns the new list of choices for the analysis field.
`nil`	Returns `nil` if the field cannot be accessed.

asiSetAnalysisFormFieldVal

```
asiSetAnalysisFormFieldVal( r_form s_analysisName s_fieldName g_value )
     => g_value | nil
```

Description

Sets the value of a field on the Choosing Analyses form.

Arguments

r_form	Form containing the analysis fields.
s_analysisName	Name of the analysis.
s_fieldName	Name of the analysis field.
g_value	Value for the analysis field.

Value Returned

g_value	Returns the new value for the analysis field.
nil	Returns nil if the field cannot be accessed.

Example

```
asiSetAnalysisFormFieldVal( form 'tran 'to "10u")
```

Sets the `to` field to 10µ for a transient analysis.

asiSetAnalysisFormWidth

```
asiSetAnalysisFormWidth( o_tool x_width )
     => x_width | nil
```

Description

Sets the width of the Choosing Analyses form. You need to add a call to this procedure to the `analysis.il` file if you do not want the inherited width of the analysis form.

Arguments

o_tool	Simulation tool object.
x_width	Width of the form, in pixels.

Value Returned

x_width	Returns *x_width* when the width is set.
`nil`	Returns `nil` otherwise.

Example

```
asiSetAnalysisFormWidth( asiGetTool( 'XYZ ) 500 )
```

Sets the width of the analysis form for the XYZ simulator to 500 pixels.

asiSetAnalysisOptionFormProperties

```
asiSetAnalysisOptionFormProperties(
     o_analysis
     [?type    s_type]
     [?width   x_width]
     [?columns x_columns]
     )
     => o_formObj | nil
```

Description

Sets the display characteristics for a new analysis options form.

Arguments

o_analysis	Analysis object.
s_type	Specifies the form type. Valid Values:
'oneD	Specifies a sequential display of fields in one column.
'twoD	Specifies a two dimensional display of fields based on row and column positions. (Use the asiAddAnalysisOption, asiChangeAnalysisOption, or asiCreateAnalysisOption functions to specify values for the rows and columns.)
'custom	Lets you specify the exact coordinate locations for each field. (Use the asiAddAnalysisOption, asiChangeAnalysisOption, or asiCreateAnalysisOption functions to specify the coordinates.)
'matrix	Specifies a matrix of equally sized fields.
	Default Value: 'oneD
x_width	Width of the form, in pixels. Default Value: The minimum default width of the form is 400

	pixels. If the fields require more space than this, the form defaults to the smallest width that can accommodate the fields.
x_columns	Number of columns. Use this argument only with matrix type forms. Default Value: 2

Value Returned

o_formObj	Returns the analysis option form object if successful.
nil	Returns nil otherwise.

Example

```
asiSetAnalysisOptionFormProperties( asiGetAnalysis( asiGetTool( 'XYZ_simulator )
'XYZ_analysis )
?width 500)
```

Sets the form properties for the analysis options for a new XYZ analysis.

asiSetAnalysisOptionChoices

```
asiSetAnalysisOptionChoices( o_analysis s_optionName l_choices )
     => l_choices | nil
```

Description

Specifies the list of choices for an analysis option that is set up as a list box.

Arguments

o_analysis	Specifies an analysis object.
s_optionName	Name of an analysis option of the type `listBox`.
l_choices	List of choices for the analysis option.

Value Returned

l_choices	Returns the new list of choices for the analysis option.
`nil`	Returns `nil` if the option does not exist.

asiSetAnalysisOptionVal

```
asiSetAnalysisOptionVal( o_analysis s_optionName g_value )
     => g_value | nil
```

Description

Sets the value of an option in an analysis.

Arguments

o_analysis	Specifies an analysis object.
s_optionName	Name of the analysis option for which you want to set the value.
g_value	Value for the analysis option.

Value Returned

g_value	Returns the new value for the analysis option.
nil	Returns nil if the option does not exist.

Example

```
asiSetAnalysisOptionVal( analysis 'lvltim "2" )
```

Sets the value of the *lvltim* analysis option to *2*.

Related Function

To display the current analysis option names, see the asiDisplayAnalysisOption function on page 9-347.

apaExport

```
apaExport(w_windowId)
```

Description

This function can be used to export the sweep specifications, specified in the given Parametric Analysis window, to a file in the comma-separated values (csv) format. When called, the function opens the *Export data in a csv format* dialog where you need to provide name of the file to which the sweep data is to be exported.

Arguments

w_windowId	The window Id of the Parametric Analysis window.

Values Returned

`t`	Always returns `t`.

Example

```
apaExport(window(5)) => t
```

apaExportCB

```
apaExportCB(w_windowId)
```

Description

Callback for the menu *File-Export to csv File* in the Parametric Analysis window.

Arguments

w_windowId	The window Id of the Parametric Analysis window.

Values Returned

`t`	Always returns `t`.

Example

```
apaExportCB(window(5)) => t
```

apaStop

```
apaStop()
```

Description

This function can be used to stop a parametric analysis. Before calling this function from the CIW, make the parametric analysis window the current window. When called, this function will terminate the current simulator run and stop the parametric analysis. Any partial data created by parametric analysis will be removed. You can use the *Pause* or *Pause Now* menus to pause the parametric analysis before calling this function.

Values Returned

`t`	Always returns `t`.

Example

```
apaStop() => t
```

apaStopCB

```
apaStopCB(w_windowId)
```

Description

Callback for the future menu *Analysis-Stop* in a parametric analysis window.

Arguments

w_windowId	The window Id of the parametric analysis window.

Values Returned

`t`	Returns `t` if the analysis was stopped.
`nil`	Returns `nil` if there was an error.

Example

```
apaStopCB(hiGetCurrentWindow())
```

10

Simulation Control Functions for Direct Interfaces

This chapter documents a set of OASIS Procedural Interfaces (PI) defined for direct simulation. Except as noted, you can redefine these methods.

We recommend that you not use or overload the following methods for OASIS direct simulation because they are either not applicable or they do not fit the direct simulation use model.

`asiSendAnalysis`
`asiSendControlStmts`
`asiSendInitCond`
`asiSendNetlist`
`asiSendNodeSets`
`asiSendOptions`
`asiSendRestore`
`asiSendDesignVars`
`asiSendKeepList`
`asiSendModelPath`
`asiSendInitFile`
`asiSendSim`
`asiSendUpdateFile`
`asiFinalNetlist`
`asiRawNetlist`
`asiAddFlowchartLink`
`asiAddFlowchartStep`
`asiCreateFlowchart`
`asiChangeFlowchartStep`
`asiDeleteFlowchartLink`
`asiDeleteFlowchartStep`
`asiExecuteFlowchart`
`asiDisplayFlowchart`
`asiGetFlowchart`
`asiInvalidateFlowchartStep`
`asiGetMarchList`
`asiSetMarchList`

The asiAnalog Class: Initialization and Simulation Control

The `asiAnalog` class is the base class for all analog simulators. A simulator derived from the `asiAnalog` class is integrated directly without using Cadence SPICE for netlisting and simulation control.

asiInitialize

```
asiInitialize( o_tool )
     => o_tool | nil
```

Description

Initializes the tools that are derived from the `asiAnalog` class. This method is not called for tools that are derived from the `asiSocket` class.

For the `asiAnalog` class, it calls:

- `asiInitStartOption`
- asiInitEnvOption
- asiInitSimOption
- asiInitFormatterClass
- asiInitAnalysis
- asiInitUI
- asiInitDataAccessFunction
- other procedures

Arguments

o_tool	The simulation session object.

Value Returned

o_tool	The simulation session object.
`nil`	Failure.

Example

```
defmethod( asiInitialize (( yourSimulator_session ))
    asiInitFormatterClass(tool)
    asiInitEnvOption(tool)
    asiInitAnalysis(tool)
```

)

asiNetlist

```
asiNetlist( o_session )
     => g_status | nil
```

Description

This method performs the creation of a design object. It then creates the formatter object with `nlCreateFormatter`, after which the netlister is run and a netlist is generated with `nlNetlist`. Netlist statistics are then printed. The netlister also provides a component count, as well as the addition of design variables found during netlisting.

This method is called by the environment, so you should not call it directly from the interface. This method can be redefined for the interface. Use `callNextMethod` in this definition.

Arguments

o_session	The OASIS session object.

Value Returned

g_status	Success.
`nil`	Netlisting errors were encountered.

Example

```
asiNetlist( session )
```

asiInterruptSim

```
asiInterruptSim( o_session )
     => g_status | nil
```

Description

This method provides an interrupt to the simulation run process for a session. It is associated with the *Simulation->Stop* action in the user interface. This method is called by the environment. Therefore, you should not call it directly from the interface. This method can be re-defined for the interface. Use *callNextMethod* in this definition.

Note for Integrators:

This function is defined as a method for the Analog Class. You can overload this method for your simulator class, as illustrated in the example. If you want to send a soft interrupt signal to your simulator, refer to the example mentioned in the *asiQuitSimulator* function.

Arguments

o_session	The OASIS session object.

Return Value

g_status	Success.
nil	Errors were encountered when interrupting the simulation.

Example

```
defmethod(  asiInterruptSim (( session yourSimulator_session ))
    println("yourSimulator : Simulation stopped by user")
    <insert code you need >
    callNextMethod()
    <insert code you need >
)
```

asiSetProjectDirChangeSetup

```
asiSetProjectDirChangeSetup( o_session )
     => t | nil
```

Description

Enables you to modify the simulator settings for the given session after changing the project directory. This method is called by the environment. Therefore, you should not call it directly from the interface.

Note for Integrators:

This function is defined as a method for the Analog Classand returns `t` at analog class. You can overload this method for your simulator class, as illustrated in the example.

Arguments

o_session	Simulation session object.

Value Returned

`t`	Returns `t` if the function call was successful.
`nil`	Returns `nil` otherwise.

Example

```
defmethod ( asiSetProjectDirChangeSetup ( session yourSimulator_session ))
    <insert code you need >
    callNextMethod()
    <insert code you need >
)
```

asiQuitSimulator

```
asiQuitSimulator( o_session [?mode g_mode] )
     => g_status | nil
```

Description

This method terminates or kills the simulator run process based on the `ipc` signal being sent to the specified process. By default, the direct integration code sends a hard-kill signal to the simulator process. To change this option and send a soft-kill signal, the *?mode* option should be set to `true` and overloaded for `yourSimulator` session. This method is called by the environment, therefore, you should not call it directly from the interface. This method can be re-defined for the interface. Use, *callNextMethod* in its definition.

Note for Integrators:

The function *asiQuitSimulator* is called internally via *asiInterruptSim*. To send a hard or soft interrupt signal, *asiInterruptSim* should be overloaded for your simulator class. This is illustrated in the example. In case you are not concerned with the hard kill scenario, it is recommended that you do not overload *asiInterruptSim* or *asiQuitSimulator* methods and use the default behavior.

Arguments

o_session	OASIS session object.
g_mode	Specifies the type of `ipc` signal sent to kill the simulator process. By default, *g_mode* is `nil`, meaning a hard-kill signal (*ipcKillProcess*) is sent to the simulator. When, *g_mode* is `t`, a soft-kill signal (*ipcSoftKill*) is sent to the simulator process. Valid Values: `nil`: a hard-kill signal (*ipcKillProcess*) is sent `t : a` soft-kill signal (*ipcSoftKill*) is sent Default Value: `nil`

Values Returned

`g_status`	Success.
`nil`	Otherwise.

Example

Note that *asiQuitSimulator* is called from *asiInterruptSim*. Therefore, a simulator session can quit based on the type of signal received as follows:

- **Hard-kill signal** (default, for `asiAnalog_session`)

In this case, you are not concerned with the `ipc` hard kill signal being sent to your simulator's session. There is no need to overload *asiQuitSimulator*. Instead, use the *callNextMethod()*, which will serve your purpose:

```
defmethod( asiInterruptSim ((session yourSimulator_session))
    println(  "yourSimulator: session  hard killed .\n" )
    println(  "yourSimulator: simulation is stopped by user.\n" )
    println(  "yourSimulator: simulation results may not be complete.\n" )
    asiSetWasInterrupted( session t )
    asiQuitSimulator( session)
)
```

- **Soft-kill signal** (customized code)

When you need to have a soft-kill signal sent to your simulator's session, you need to overload *asiQuitSimulator* method with *?mode* set to `t`, as follows:

```
defmethod( asiInterruptSim ((session yourSimulator_session))
    <insert code you need>
    println(  "yourSimulator: session  soft killed .\n" )
    println(  "yourSimulator: simulation is stopped by user.\n" )
    println(  "yourSimulator: simulation results may not be complete.\n" )
    asiSetWasInterrupted( session t )
    asiQuitSimulator( session ?mode t )
    <insert code you need>
)
```

Integrator Overloadable Methods for Invoking Simulation

asiRunSimulation

```
asiRunSimulation( o_session )
     => g_status | nil
```

Description

This method performs the simulation for the session. This method is called by the environment, so you should not call it directly from the interface. This method can be redefined for the interface. Use `callNextMethod` in this definition.

Arguments

o_session	The OASIS session object.

Value Returned

g_status	Success.
`nil`	Errors were encountered when starting the simulation.

Example

```
defmethod( asiRunSimulation (( session yourSimulator_session))
    <insert code you need >
    callNextMethod()
    <insert code you need>
```

asiGetPredefinedCommandLineOption

```
asiGetPredefinedCommandLineOption( o_session )
     => t_predefinedCmdLineOption
```

Description

Gets the predefined simulation command line options. This function returns an empty string at the `asiAnalog` class. Overload this method for your simulator.

Arguments

o_session	The simulation session object.

Value Returned

t_predefinedCmdLineOption
The predefined command line options in a string.

Example

```
defmethod( asiGetPredefinedCommandLineOption (
                        ( _session yourSimulator_session ))
"any command line args that you always send to your simulator"
)
```

asiGetCommandFooter

```
asiGetCommandFooter( o_session )
     => t_commandFooter
```

Description

Specifies the footer of the simulation run command.

Argument

o_session: Simulation session object.

Value Returned

t_commandFooter
Command line footer.

Example

```
defmethod( asiGetCommandFooter (( session mySimulator_session ))
    " > mySimulator.log"
)
```

Integrator Overloadable Methods for Formatting Control Statements

asiFormatControlStmts

```
asiFormatControlStmts( o_session p_fp)
     => t | nil
```

Description

Creates and formats all control statements. It formats the following in the following order: node sets by calling `asiFormatNoteSet`; initial conditions by calling `asiFormatInitCond`; simulator options by calling `asiFormatSimulatorOptions`; analyses by calling `asiFormatAnalysisList`; nets/currents to save by calling `asiFormatKeepList`.

Arguments

o_session	The simulation session object.
p_fp	The file pointer.

Value Returned

`t`	The complete simulator input file was generated successfully.
`nil`	Otherwise.

Example

```
defmethod( asiFormatControlStmts (( session asiAnalog_session ) fp )
callNextMethod()
asiFormatMyControlStmt( session fp )
)
```

asiFormatDesignVarList

```
asiFormatDesignVarList( o_session )
     => t | nil
```

Description

Formats and prints the design variable statements to the design variable file. This routine first prints the string `.PARAM` followed by the design variables in name=value pairs. The design variables are obtained by calling `asiGetDesignVarList`.

Arguments

o_session	The simulation session object.

Value Returned

`t`	The design variable statements were generated successfully.
`nil`	Otherwise.

Example

```
defmethod( asiFormatDesignVarList ((session asiAnalog_session) fp )
let( ( (varList asiGetDesignVarList( session ))
         mappedVarList mappedVar)
    when( varList
      mappedVarList = asiGetDesignVarMappedList( session )
     artFprintf( fp ".PARAM")
     foreach( var varList
        mappedVar = assoc( car(var) mappedVarList)
        if( mappedVar
            artFprintf( fp " %s" (cadr mappedVar ))
            artFprintf( fp " %s" car( var ))
         )
        unless( (artBlankString (cadr var))
            artFprintf( fp "=%s" (cadr var)))
        )
        artFprintf( fp "\n" )
      )
      t
```

```
    )
)
```

asiFormatInitCond

```
asiFormatInitCond( o_session p_fp )
     => t | nil
```

Description

Formats and prints the initial condition commands to the control statement file. This routine prints the string `.IC` followed by the initial conditions in V(net)=voltage pairs.

Arguments

o_session	The simulation session object.
p_fp	The pointer to the control statement file.

Value Returned

`t`	The initial condition commands were formatted successfully.
`nil`	Otherwise.

Example

```
defmethod( asiFormatInitCond (( session asiAnalog_session )  fp )
    let( ( (icList asiGetInitCondList( session ))
           ( netlistDir asiGetAnalogNetlistDir( session ) ))
        when( icList
            artFprintf(fp ".IC ")
            foreach( obj icList
                foreach( name
                    asiMapOutputName( netlistDir asiGetSelObjType( obj )
                                      asiGetSelObjName( obj ) )
                    artFprintf(fp "V( %s )=%s" name asiGetSelObjValue(obj))
                )
            )
            artFprintf(fp "\n")
        )
        t
    )
)
```

asiFormatNodeSet

```
asiFormatNodeSet (o_session p_fp )
     => t | nil
```

Description

Formats and prints the nodeset commands to the control statement file. This routine prints `.NODESET` and then the nodesets in V(net)=voltage pairs.

Arguments

o_session	The simulation session object.
f_fp	The pointer to the control statement file.

Value Returned

`t`	The nodeset commands were formatted successfully.
`nil`	Otherwise.

Example

```
defmethod( asiFormatNodeSet ( ( session asiAnalog_session ) fp )
    let( ( (nodeSetList asiGetNodeSetList( session ))
           ( netlistDir asiGetAnalogNetlistDir( session ) ))
        when( nodeSetList
            artFprintf(fp ".NODESET ")
            foreach( obj nodeSetList
                foreach( name
                    asiMapOutputName( netlistDir asiGetSelObjType( obj )
                                      asiGetSelObjName( obj ) )
                    artFprintf(fp "V( %s )=%s" name
                                   asiGetSelObjValue(obj))
                )
            )
            artFprintf(fp "\n")
        )
        t
    )
```

)

asiFormatKeepList

```
asiFormatKeepList( o_session p_fp )
     => t | nil
```

Description

Formats and prints the signal save commands to the control statement file. At the `asiAnalog` class this routine simply returns `t`. You need to create your own `asiFormatKeepList` routine.

Arguments

o_session	The simulation session object.
f_fp	The pointer to the control statement file.

Value Returned

`t`	The save commands were formatted successfully.
`nil`	Otherwise.

Example

```
defmethod( asiFormatKeepList ( ( session spectre_session ) fp )
   let( ((netlistDir (asiGetAnalogNetlistDir session))
         (keepList (asiGetKeepList session))
         nameList )
        when( keepList
             artFprintf(fp "save ")
             foreach( keep keepList
                 when( memq( asiGetSelObjType( keep ) '( net terminal ))
                     nameList = asiMapOutputName( netlistDir
                         asiGetSelObjType( keep ) asiGetSelObjName( keep ))
                     foreach( name nameList
                         artFprintf( "%s " name )
                     )
                 )
             )
             artFprintf(fp "\n")
```

```
            )
            t
        )
)
```

asiFormatSimulatorOptions

```
asiFormatSimulatorOptions( o_session p_fp )
     => t | nil
```

Description

Formats and prints the simulator option statements to the designated file. This routine prints `.OPTIONS` followed by name=value pairs.

Arguments

o_session	The simulation session object.
f_fp	The pointer to the control statement file.

Value Returned

`t`	The simulator options were formatted successfully.
`nil`	Otherwise.

Example

```
defmethod( asiFormatSimulatorOptions
            (( session asiAnalog_session ) fp )
    let(( value (firstOption t))
        foreach( option asiGetSimOptionList( session )
            when( value = asiGetFormattedVal( option )
                when( firstOption
                    artFprintf(fp ".OPTIONS ")
                    firstOption=nil
                )
                artFprintf(fp "%s=%s " asiGetName(option) value)

            )
        )
        t
    )
)
```

asiFormatAnalysisList

```
asiFormatAnalysisList( o_ana p_fp)
     => t | nil
```

Description

Formats all enabled analyses by calling `asiFormatAnalysis`.

Arguments

o_ana	The analysis object.
p_fp	The pointer to the control statement file.

Value Returned

`t`	All enabled analyses were formatted successfully.
`nil`	Otherwise.

Example

```
defmethod(asiFormatAnalysisList ((session yourSimulator_session) fp)
;; add your code
callNextMethod()
;; add your code
t
)
```

asiFormatAnalysis

```
asiFormatAnalysis( o_ana p_fp)
     => t | nil
```

Description

Formats and prints analysis statements to the control file. For the general asiAnalog class, it follows this routine: prints the analysis name by calling `asiGetAnalysisName`; prints the list of signals by calling `asiGetAnalysisSigList` and formats them in parentheses [for example: `(net1 net2)`]; prints the analysis field list in name=value pairs; prints the analysis options in name=value pairs; uses `asiGetFormattedVal()` to obtain the print string for an analysis field value or an analysis option value. Please see the description of `asiGetFormattedVal` routine for more details.

Arguments

o_ana	The analysis object.
p_fp	The pointer to the control statement file.

Value Returned

`t`	The analysis statements were formatted successfully.
`nil`	Otherwise.

Example

```
defmethod( asiFormatAnalysis ((ana asiAnalog_analysis) fp)
    let(( name type sigList simVal (session asiGetSession(ana) ))
        name = asiGetAnalysisName( ana )
        sigList = asiGetAnalysisSigList( session ana )
        ;;; prints analysis name
        artFprintf(fp "%s " name )

        ;;; handls analysis signals
        when( sigList
            artFprintf( fp "( ")
            foreach( netField sigList
                when( simVal = asiGetFormattedVal( netField )
```

```
                    artFprintf(fp "%s " simVal)
                )
            )
            artFprintf( fp ") ")
        )



        ;;; prints analysis fields
        foreach( f asiGetAnalysisSimFieldList( session ana )
            when( simVal = asiGetFormattedVal( f )
                artFprintf(fp "%s=%s " asiGetName(f) simVal)
            )
        )

        ;;; prints analysis options
        foreach( o asiGetAnalysisOptionList( ana )
            when( simVal = asiGetFormattedVal(o)
                artFprintf(fp "%s=%s " asiGetName(o) simVal)
            )
        )

        artFprintf(fp "\n")
        t
    )
)
```

asiFormatModelLibSelectionList

```
asiFormatModelLibSelectionList( o_session p_fp)
     => t | nil
```

Description

Formats the statement which specifies the model library information.

Arguments

o_session	The simulation session object.
p_fp	The pointer to the control statement file.

Value Returned

t	All model library selections were formatted successfully.
nil	Otherwise.

Example

The generic method for asiAnalog class does nothing. The simulator interface has to define its own method. The following is an example:

```
defmethod(asiFormatAnalysisList ((session xyz_session) fp)
foreach( obj asiGetModelLibSelectionList( session )
    artFprintf( fp "include %L" asiGetModelLibFile(obj))
    unless( artBlankString(section)
        artFprintf( fp " section=%s" asiGetModelLibSection(obj))
        )
        artFprintf( fp "\n")
    ) ; foreach
t )
```

asiFormatDefinitionFileList

```
asiFormatDefinitionFileList( o_session p_fp)
     => t | nil
```

Description

Formats the statement which includes the specified definition files.

Arguments

o_session	The simulation session object.
p_fp	The pointer to the control statement file.

Value Returned

t	All definition files were formatted successfully.
nil	Otherwise.

Example

```
defmethod( asiFormatDefinitionFileList (( session xyz_session ) fp )
  foreach( d asiGetDefinitionFileList(session)
    artFprintf( fp "include '%s'\n" d )
  )
)
```

asiFormatTextStimulusFileList

```
asiFormatTextFileList( o_session p_fp)
     => t | nil
```

Description

Formats the statement which includes the textual stimulus files.

Arguments

o_session	The simulation session object.
p_fp	The pointer to the control statement file.

Value Returned

`t`	All textual stimulus files were formatted successfully.
`nil`	Otherwise.

Example

```
defmethod(asiFormatTextStimulusFileList (( session xyz_session ) fp )
  foreach( d asiGetTextStimulusFileList(session)
    artFprintf( fp "include '%s'\n" d )
  )
)
```

asiNeedSuffixEvaluation

```
asiNeedSuffixEvaluation( o_session )
     => t | nil
```

Description

Specify whether the interface needs suffix evaluation or not. When this method returns t, the numeric suffixes specified in a numericString field will be evaluated. For example, suppose the start frequency field for the AC analysis has a value of 10M, asiGetFormattedVal(ac_start_fieldObj) returns 1e7 provided it is created as a ‘numericString field.

Arguments

o_session	The simulation session object.

Value Returned

t	The Numeric suffix needs to be evaluated by the environment
nil	Otherwise.

Example

```
defmethod(asiNeedSuffixEvaluation (( session xyz_session ) fp )
  nil
)
```

In this case, the numeric suffix will not be evaluated

asiInvalidateControlStmts

```
asiInvalidateControlStmts ( {o_session | o_tool} )
     => t | nil
```

Description

The `asiInvalidateControlStmts` function is a wrapper to `asiInvalidateFlowchartStep`, which invalidates the `asiSendControlStmts` flowchart step.

Arguments

o_session	Simulation session object.
o_tool	Simulation tool object.

Values Returned

`t`	Invalidates the `asiSendControlStmts` flowchart step
`nil`	Otherwise

Example

```
defmethod( asiInvalidateControlStmts ( ( session spectreS_session ) )
asiInvalidateFlowchartStep( session 'asiSendControlStmts )
)
```

Utility Functions

Do not overload the utility functions described in this section.

asiGetSimExecName

```
asiGetSimExecName( o_session )
     => t_simulatorExecutableName
```

Description

Gets the name of the simulator executable by calling `asiGetSimName`.

Arguments.

o_session	The simulation session object.

Value Returned

t_simulatorExecutableName	The simulator executable name.

Example

```
defmethod( asiGetSimExecName (( yourSimulator_session ))
    "your simulator executable name"
)
```

asiGetCommandLineOption

```
asiGetCommandLineOption( o_session )
     => t_CommandLineOption
```

Description

Gets the simulation command line options. At the `asiAnalog` class this method returns the value of the environment option `'userComdLineOption`.

Arguments

o_session	The simulation session object.

Value Returned

t_CommandLineOption	The command line options in a string.

Example

```
defmethod( asiGetCommandLineOption (
                    ( _session yourSimulator_session ))
strcat( asiGetPredefinedCommandLineOption( session )
        asiGetEnvOptionVal( session 'userCmdLineOption )
    )
)
```

asiGetAnalysisSigList

```
asiGetAnalysisSigList( o_session o_ana )
     => l_sigObjList
```

Description

Gets a list of analysis field objects which are of the type net. For example, the p and n nodes for the Spectre noise analysis.

Arguments

o_session	The simulation session object.
o_ana	The analysis object.

Value Returned

l_sigObjList	A list of analysis signal objects.

Example

```
asiGetAnalysisSigList( session ana )
```

asiGetAnalysisType

```
asiGetAnalysisType( o_analysis )
     => s_analysisType
```

Description

Gets the type of the analysis.

Argument

o_analysis	Specifies an analysis object

Value Returned

s_analysisType	The type of the analysis

Example

```
asiGetAnalysisType( analysis )
```

asiGetAnalysisSimFieldList

```
asiGetAnalysisSimFieldList( o_session o_ana )
     => l_simFieldObjList
```

Description

Gets a list of simulator analysis field objects which need to be netlisted.

Arguments

o_session	The simulation session object.
o_ana	The analysis object.

Value Returned

l_simFieldObjList	The list of simulator field objects.

Example

```
asiGetAnalysisSimFieldList( session ana )
```

asiGetModelLibSelectionList

```
asiGetModelLibSelectionList( o_session )
     => l_modelLibSelectionList | nil
```

Description

Formats the statement which specifies the model library information.

Arguments

o_session	The simulation session object.

Value Returned

`l_modelLibSelectionList`	The list of model library selection objects.
`nil`	Otherwise.

Example

The following example shows how to get the list of model file names in the current ADE L session:

```
session = asiGetCurrentSession()
=> stdobj@0x235ff080

modelList = asiGetModelLibSelectionList( session )
=> (("./Models/myModels.scs" "FF")
("./Models/InlineModels.scs" "")
("./Models/mySingle.scs" "fastfast")
)
;The following statement returns the model file name for the first model library
in the current ADE L session
asiGetModelLibFile(car(modelList))
=> "./Models/myModels.scs"
```

asiGetModelLibFile

```
asiGetModelLibFile( o_modelLibSelection )
     => t_fileName | nil
```

Description

Gets the file name of a model library selection object.

Arguments

o_modelLibSelection The model library selection object.

Value Returned

t_fileName	The model file name.
nil	Otherwise.

Example

The following example shows how to get the model file names for the first model library in the current ADE L session:

```
session = asiGetCurrentSession()
=> stdobj@0x235ff080

modelList = asiGetModelLibSelectionList( session )
=> (("./Models/myModels.scs" "FF")
("./Models/InlineModels.scs" "")
("./Models/mySingle.scs" "fastfast")
)

asiGetModelLibFile(car(modelList))
=> "./Models/myModels.scs"
```

asiGetModelLibSection

```
asiGetModelLibSection( o_modelLibSelection )
     => t_sectionName | nil
```

Description

Gets the section name of a model library selection object.

Arguments

o_modelLibSelection The model library selection object.

Value Returned

t_sectionName	The section name within a model library file.
nil	Otherwise.

Example

The following example shows how to get the section name for the first model library in the current ADE L session:

```
session = asiGetCurrentSession()
=> stdobj@0x235ff080

modelList = asiGetModelLibSelectionList( session )
=> (("./Models/myModels.scs" "FF")
("./Models/InlineModels.scs" "")
("./Models/mySingle.scs" "fastfast")
)

asiGetModelLibSection(car(modelList))
=> "FF"
```

asiGetDefinitionFileList

```
asiGetDefinitionFileList( o_session )
     => l_definitionFileList | nil
```

Description

Gets the list of definition file names associated with the given simulation session.

Arguments

o_session	The simulation session object.

Value Returned

l_definitionFileList	The list of definition file names.
nil	Otherwise.

Example

```
asiGetDefinitionFileList( session )
```

asiGetTextStimulusFileList

```
asiGetDefinitionFileList( o_session )
     => l_definitionFileList | nil
```

Description

Gets the list of textual stimulus file names associated with the given simulation session.

Arguments

o_session	The simulation session object.

Value Returned

l_stimFileList	The list of textual stimulus file names.
nil	Otherwise.

Example

```
asiGetTextStimulusFileList( session )
```

asiGetFormattedVal

```
asiGetFormattedVal( o_anaField | o_anaOption | o_simOption ) =>t_formattedVal
     / nil
```

Description

Gets the string value of an analysis field object, an analysis option object, or a simulator option object. The format of the string values are based on the object types which are specified with functions such as `asiCreateAnalysisField`, `asiCreateAnalysisOption`, `asiAddSimOption`. A number of examples can be found in the example section.

Arguments

o_anaField:	The analysis field object.
o_anaOption:	The analysis option object.
o_simOption:	The simulator option object.

Value Returned

t_formattedVal	The formatted simulator value.
`nil`	An object should not be printed in the control statement file.

Example

```
option = asiGetSimOptionList(session)
value = asiGetFormattedVal(option)
```

For example, if a simulator option of the type '`literalString`' is defined by :

```
asiAddSimOption( tool
    ?name 'option
    ?type 'literalString
    ?value "value1"
)
```

`asiGetFormattedVal()` for this option returns

```
"\"value1\""
```

When the simulator option is defined by:,

```
asiAddSimOption( tool
    ?name 'option
    ?type 'literalString
    ?value "value1"
    ?literalStartMarker "'"
    ?literalEndMarker  "'"
)
```

`asiGetFormattedVal()` for this option returns

```
"'value1'"
```

The following is an example of printing out a list:

```
asiAddSimOption( tool
    ?name 'option
    ?type 'list
    ?value '( 1 2 3 )
    ?startMarker "["
    ?endMarker  "]" )

asiGetFormattedVal() for this option returns
[ 1 2 3 ]
```

The following example illustrate the numericString type formatting in relation to the Value Returned of asiNeedSuffixEvaluation method.

Given that a simulator option is defined by:

```
asiAddSimOption( tool
    ?name ‘option1
    ?type ‘numericString
    ?value “1M”
)
When asiNeedSuffixEvaluation method for xyz tool returns t then
asiGetFormattedVal() for this option returns “1e6”.
Whan asiNeedSuffixEvaluation method for xyz tool returns nil then
asiGetFormattedVal(0 for this option returns “1M”.
```

asiGetSelObjName

```
asiGetSelObjName( o_selObj )
     => t_name
```

Description

Returns the schematic name of the selected signal object.

Arguments

o_selObj	Selected signal object. Initial condition, nodeset, and keep objects are all selection objects.

Value Returned

t_name	Name of the selected signal object.

Example

An example usage of this routine can be found in the example of `asiFormatInitCond`.

asiGetSelObjType

```
asiGetSelObjType( o_selObj )
     => t_signalType
```

Description

Returns the type of the selected signal object.

Arguments

o_selObj	Selected signal object. Initial condition, nodeset, and keep objects are all signal objects

Value Returned

t_signalType	Name of the selected signal object. The possible values are `net, `terminal, and `instance.

Example

An example usage of this routine can be found in the example of asiFormatInitCond.

asiGetSelObjValue

```
asiGetSelObjValue( o_selObj ) => t_value
```

Description

Returns the initial condition or nodeset values specified on the selected signal object.

Arguments

o_selObj	Selected signal object. Initial condition, nodeset, and keep objects are all signal objects

Value Returned

t_value	The initial condition value or nodeset voltage value specified on the selected signal object.

Example

An example usage of this routine can be found in the example of `asiFormatInitCond`.

asiMapOutputName

```
asiMapOutputName( t_dir s_type t_name @key formatflag s_formatflag)
     => l_nameList
```

Description

Maps the given schematic name of the given type using the netlist directory. The result is a list of mapped strings.

Arguments

t_dir	Netlist directory.
s_type	Type of object. Valid types are `'net`, `'instance` and `'terminal`.
t_name	Schematic name.
s_formatflag	Performs name mapping of transient simulation data available in SST2 format. Set the value to `"t"` if this function is used in context of transient simulation data analysis and the transient simulation data is available in SST2 format.

Value Returned

l_nameList	List of mapped names.
`nil`	On failure.

Example

```
asiMapOutputName( "netlistDir" 'net "/net13" )
```

Another example of the usage of this routine can be found in the example section of `asiFormatInitCond`.

asiGetSimInputFileList

```
asiGetSimInputFileList( o_session )=> l_fileNamesList
```

Description

Returns a list of all file names concatenated to generate the input file to the simulator. You can override this method to add/delete files used to generate the final input file to your simulator.

Arguments

o_session	Specifies a simulation session object.

Value Returned

l_fileNames	List list of file names used to generate input file to the simulator

Example

```
defmethod( asiGetSimInputFileList (( session <yourSimulator>_session>))
    cons( "veriloga.inc" callNextMethod() )
)
```

Adds the contents of file `veriloga.inc` to the final input file to the simulator.

artInvalidateAmap

```
artInvalidateAmap( )=> t | nil
```

Description

Resets the in-memory Amap cache. Further mapping function calls will result in re-reading the amap files from disk.

Arguments

None

Value Returned

t	It returns *t* if the call is successful.
nil	It returns *nil* if the call is unsuccessful.

11

Flowchart Functions

This chapter describes the functions that modify the simulation flowchart for socket and direct interfaces. You can modify the flowchart for the direct interface, if required. The flowchart describes the tool flow (the order in which certain events are expected to occur).

asiAddFlowchartLink

```
asiAddFlowchartLink( o_flowchart s_parentStep s_childStep )
     => t | nil
```

Description

Creates a new link between the specified parent and child steps, which were created with `asiAddFlowchartStep`.

Arguments

o_flowchart	Flowchart object.
s_parentStep	Name of the parent step.
s_childStep	Name of the child step.

Value Returned

`t`	Returns `t` if the link is created.
`nil`	Returns an error message and `nil` otherwise.

Example

```
flowchart = asiGetFlowchart( session )
asiAddFlowchartLink( flowchart 'asiRawNetlist
    'asiRunSimulation )
```

Adds a flowchart link between the netlist and simulate steps.

asiAddFlowchartStep

```
asiAddFlowchartStep(
     o_flowchart
     ?name     s_name
     [?description t_description]
     [?runMessage t_runMessage]
     [?function s_function]
     [?checkFunc s_checkFunc]
     [?preFunc s_preFunc]
     [?postFunc s_postFunc]
     [?ignoreFunc s_ignoreFunc]
     )
     => o_step | nil
```

Description

Adds a new step to an existing flowchart.

Arguments

o_flowchart	Flowchart object to which you want to add a step.
s_name	Name of the step to add.
t_description	Textual description of the step.
t_runMessage	Message to print when this step is executed.
s_function	The (primary) procedure to call to execute this step. Callback parameter list: (*o_session*)
s_checkFunc	Function to evaluate to update the status of this step. This function is executed after the dependency requirement is met. Callback parameter list: (*o_session*)
s_preFunc	Function to evaluate immediately before the primary function is executed. It can be added to a step instance to customize an existing step. Callback parameter list: (*o_session*)
s_postFunc	Procedure to execute after *s_function*. Callback parameter list: (*o_session*)

s_ignoreFunc	Function to skip.

Value Returned

o_step	Returns the flowchart step object.
nil	Returns nil if there is an error.

Example

```
asiAddFlowchartStep( flowchart
?name 'sendABCfile
?description "Send the ABC file"
?runMessage "sending ABC file..."
?function 'sendABCfile
)
```

For the XYZ simulator, add a new step to send the ABC file to Cadence SPICE.

asiChangeFlowchartStep

```
asiChangeFlowchartStep(
     o_flowchart
     ?name     s_name
     [?descriptiont_description]
     [?runMessaget_runMessage]
     [?functions_function]
     [?checkFuncs_checkFunc]
     [?preFunc s_preFunc]
     [?postFuncs_postFunc]
     )
     => o_step | nil
```

Description

Changes a flowchart step in an existing flowchart

Arguments

o_flowchart	Flowchart object with the step you want to change.
s_name	Name of the flowchart step to change.
t_description	Description of the step.
t_runMessage	Message to print when this step is executed.
s_function	The (primary) procedure to call to execute this step. Callback parameter list: (*o_session*)
s_checkFunc	Procedure that is executed during each traversal of the flowchart. This procedure is used to set the status of the steps. Callback parameter list: (*o_session*)
s_preFunc	Procedure to execute before *s_function*. Callback parameter list: (*o_session*)
s_postFunc	Procedure to execute after *s_function*. Callback parameter list: (*o_session*)

Value Returned

o_step	Returns the new step object.
nil	Returns nil if the specified step does not exist.

Example

```
asiChangeFlowchartStep( o_flowchart
    ?name sendXYZFile
    ?runMessage "Sending XYZ file"
)
```

Changes the run message for the *sendXYZFile* flowchart step.

Related Function

To display the current flowchart steps and links, see the asiDisplayFlowchart function on page-440.

asiCreateFlowchart

```
asiCreateFlowchart( o_tool )
     => o_flowchart
```

Description

Creates a new flowchart.

Arguments

o_tool	Simulation tool object.

Value Returned

o_flowchart	Returns the flowchart object.

asiDeleteFlowchartLink

```
asiDeleteFlowchartLink( o_flowchart s_parentStep s_childStep )
     => t | nil
```

Description

Deletes the link between the specified parent and child steps.

Arguments

o_flowchart	Flowchart object.
s_parentStep	Name of the parent step.
s_childStep	Name of the child step.

Value Returned

`t`	Returns `t` if the link between the steps is removed.
`nil`	Returns `nil` otherwise.

Example

```
asiDeleteFlowchartLink( flowchart
'asiSendControlStmts 'asiRunSimulation
)
```

Removes the link connecting the `asiSendControlStmts` step to the `asiRunSimulation` step.

Related Function

To display the current flowchart steps and links, see the asiDisplayFlowchart function on page-440.

asiDeleteFlowchartStep

```
asiDeleteFlowchartStep( o_flowchart s_name [s_splice] )
     => t | nil
```

Description

Deletes a step and any attached links from an existing flowchart. Typically, you do not need this function because you can *unlink* any flowchart step that you do not want to use with the `asiDeleteFlowchartLink` function.

Arguments

o_flowchart	Flowchart object with the step you want to delete.
s_name	Name of the step to delete.
s_splice	Specifies whether the parents of the deleted step are to be linked directly to the children of the deleted step. Valid Values: t specifies that the new link is to be created, nil specifies that the new link is not to be created

Value Returned

`t`	Returns `t` if the step is deleted.
`nil`	Returns `nil` otherwise.

Example

```
flowchart = asiGetFlowchart( asiGetTool( 'XYZ ))
asiDeleteFlowchartStep( flowchart 'asiSendOptions )
```

Deletes the `asiSendOptions` step from flowchart for the XYZ simulation.

Related Function

To display the current flowchart steps and links, see the asiDisplayFlowchart function on page-440.

To delete a flowchart link, see the asiDeleteFlowchartLink function on page-438.

asiDisplayFlowchart

```
asiDisplayFlowchart( o_tool [s_rootstep] )
     => t | nil
```

Description

Displays the current steps and links for the flowchart. You can display all the step and link information, or you can display the steps and links that are below the *rootstep* step. Use this function only to determine which part of the flowchart you want to modify. Do not use this function as part of another procedure.

Arguments

o_tool	Simulation tool object.
s_rootstep	Name of the highest step (in the flowchart) to be displayed. The function returns this step and all the links and steps below it. If you do not specify this step, all the flowchart steps are displayed.

Value Returned

`t`	Returns `t` and displays a list of the current steps and links for the flowchart.
`nil`	Returns `nil` otherwise.

Example

```
asiDisplayFlowchart( asiGetTool( 'analog ) )
```

Returns the flowchart description for the *analog* tool.

asiExecuteFlowchart

```
asiExecuteFlowchart( o_session s_goalStep
     [s_printMessages] )
     => t | nil
```

Description

Executes the flowchart for a given session up to and including the goal step.

Arguments

o_session	Simulation session object.
s_goalStep	Name of the last flowchart step to execute.
s_printMessages	Specifies whether or not the *runmessages* for the steps are suppressed as they are executed. Valid Values: `t` displays the *runmessages*, `nil` suppresses the *runmessages* Default Value: `t`

Value Returned

`t`	Returns `t` if the flowchart executes as far as the goal step.
`nil`	Returns `nil` if errors are encountered.

Example

```
asiExecuteFlowchart( session 'asiRawNetlist )
```

Executes the flowchart as far as the raw netlisting step.

asiFinalNetlist

```
asiFinalNetlist( o_session )
     => t | l_dpl
```

Description

Creates the final netlist.

Note for Integrators: This function is defined as a method for the Analog Class. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_session	Specifies the session object.

Value Returned

t	Returns t when the final netlist is created (for analog only).
l_dpl	Returns a disembodied property list with the following properties (for mixed-signal only):

netlistDir	Name of the netlist directory.
finalFileList	List of final netlist files.

Example for Integrators

```
defmethod( asiFinalNetlist (( session <yourSimulator>_session ))
    <insert any code you need>
)
```

asiGetFlowchart

```
asiGetFlowchart( { o_tool | o_session } )
     => o_flowchart
```

Description

Gets the flowchart object for a tool or session.

Arguments

o_tool	Simulation tool object.
o_session	Simulation session object.

Value Returned

o_flowchart	Returns the flowchart object.

Example

```
flowchart = asiGetFlowchart( asiGetTool('analog) )
```

Returns the flowchart object for the *analog* tool.

asiInit<yourSimulator>Flowchart

```
asiInit<yourSimulator>Flowchart( o_tool )
     => t
```

Description

Calls the procedures to initialize the flowchart for your simulator.

Note: You must write `asiInit<yourSimulator>Flowchart`, where *<yourSimulator>* is the name of your simulator. Do not include the angle brackets (<>).

Arguments

o_tool	Simulation tool object.

Value Returned

`t`	Returns `t` when your procedures are called.

Note: Write `asiInit<yourSimulator>Flowchart` to return `t`.

Example

The following procedure initializes the flowchart for the XYZ simulator. It adds the step to send the library path, then inserts that step between the step to send the update file and the step to send the control statements.

```
procedure( asiInitXYZFlowchart( tool )
    let( ( flowchart )

        flowchart = asiGetFlowchart( session )

        ;;; SEND THE LIB PATH
        asiAddFlowchartStep( flowchart
            ?name               'asiSendLibPath
            ?description        "Sends the library path"
            ?runMessage         "send lib path"
            ?function           'asiSendLibPath
        )


        asiDeleteFlowchartLink( flowchart 'asiSendUpdateFile
            'asiSendControlStmts )
        asiAddFlowchartLink(    flowchart 'asiSendUpdateFile
            'asiSendLibPath )
        asiAddFlowchartLink(    flowchart 'asiSendLibPath
            'asiSendControlStmts )
```

```
        t
    )
)
```

Note: It is preferable to include the code for sending the `lib` path in the `asiSendControlStmts` step if possible. In other words, do not add extra links if they are not needed. (See the asiSendControlStmts function on page-449 for an example.)

asiInvalidateFlowchartStep

```
asiInvalidateFlowchartStep( o_session s_step )
     => t | nil
```

Description

Invalidates a flowchart step for a particular session.

You can use this function to invalidate a particular step that has become obsolete or out of date. This way, when a step that is dependent on the invalidated step must be executed, the system will first re-execute the invalidated to step to make it current.

Arguments

o_session	Simulation session object.
s_step	Name of the flowchart step that you want to invalidate.

Value Returned

`t`	Returns `t` if the flowchart step is invalidated.
`nil`	Returns `nil` otherwise.

Example

```
asiInvalidateFlowchartStep( session 'asiRawNetlist )
```

Invalidates the netlisting step for the session.

Related Function

To display the current flowchart steps and links, see the asiDisplayFlowchart function on page-440.

asiRawNetlist

```
asiRawNetlist( o_session )
     => t | nil
```

Description

Creates a raw netlist.

Note for Integrators: This function is defined as a method for the Analog Class. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_session	Specifies the session object.

Value Returned

`t`	Returns `t` if the raw netlist is created.
`nil`	Returns `nil` otherwise.

Example for Integrators

```
defmethod( asiRawNetlist (( session <yourSimulator>_session ))
    <insert any code you need>
)
```

asiSendAnalysis

```
asiSendAnalysis( o_session )
     => t | nil
```

Description

Sends analyses to Cadence SPICE by calling `asiFormatAnalysis` for each analysis.

Note for Integrators: This function is defined as a method for the Analog Class and is called by `asiSendControlStmts`. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_session	Specifies the session object.

Value Returned

`t`	Returns `t` when the analyses are sent to Cadence SPICE.
`nil`	Returns `nil` if there is an error.

Example for Integrators

```
defmethod( asiSendAnalysis (( session <yourSimulator>_session ))
    <insert any code you need>
)
```

asiSendControlStmts

```
asiSendControlStmts( o_session )
     => t | nil
```

Description

Sends information such as nodesets, initial conditions, keep lists or output, analyses, restore files, include files, and stimulus files to Cadence SPICE.

Note for Integrators: This function is defined as a method for the Analog Class. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_session	Specifies the session object.

Value Returned

t	Returns t when the different commands are sent to Cadence SPICE.
nil	Returns nil if there is an error.

Example for Integrators

```
defmethod( asiSendControlStmts (( session <yourSimulator>_session ))

    <insert any code you need>

)
```

Note: You might want to send more information for your simulator. For example, you might send the information to send the lib path to your simulator. In this case, you can use callNextMethod, as shown in the following example:

```
defmethod( asiSendControlStmts (( session <yourSimulator>_session ))

    ;Use callNextMethod to send nodesets, intitial
    ;conditions, force nodes, keepLists, and analyses

    callNextMethod()
    ;Now call your routine to send the lib path

    <yourSimulator>SendLib( session )
)
```

asiSendDesignVars

```
asiSendDesignVars( o_session )
     => t | nil
```

Description

Sends the design variables to Cadence SPICE.

Note for Integrators: This function is defined as a method for the Analog Class. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_session	Specifies the session object.

Value Returned

t	Returns t if the design variables are sent to Cadence SPICE.
nil	Returns nil if there is an error.

Example for Integrators

```
defmethod( asiSendDesignVars (( session <yourSimulator>_session ))
<insert any code you need>
)
```

asiSendInitCond

```
asiSendInitCond( o_session )
     => t | nil
```

Description

Places all the initial conditions in <*netlistDirectory*>/raw/ics and sends a ptprop command to Cadence SPICE.

There is a mechanism in the simdot.f file that gets the value of the ptprop command (if the ptprop command was sent). If the ptprop command was sent, the ics file is automatically included in the final netlist. If the ptprop command was not sent, the ics file is not included. The initial conditions are of the form:

```
ic v(node)=value
```

Note for Integrators: This function is defined as a method for the Analog Class and is called by asiSendControlStmts. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_session	Specifies the session object.

Value Returned

t	Returns t when the initial conditions are sent to Cadence SPICE.
nil	Returns nil if there is an error.

Example for Integrators

```
defmethod( asiSendInitCond(( session <yourSimulator>_session ) )
    let( ( icStr )

        icStr = ".ic v(%s) = %s\n"
        asiiSendInitCond( session icStr )

        t
    )
)
```

asiSendInitFile

```
asiSendInitFile( o_session )
     => t | nil
```

Description

Sends the `init.s` file to Cadence SPICE.

Note for Integrators: This function is defined as a method for the Analog Class. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_session	Specifies the session object.

Value Returned

`t`	Returns `t` when the initialization file is sent to Cadence SPICE.
`nil`	Returns `nil` if there is an error.

Example for Integrators

```
defmethod( asiSendInitFile (( session <yourSimulator>_session ))
<insert any code you need>
)
```

asiSendKeepList

```
asiSendKeepList( o_session )
     => t | nil
```

Description

Sends the keep list to Cadence SPICE. The keep list can contain a list of nets or currents to save and it can contain statements to *keep all nets* or *keep all currents*.

Note for Integrators: This function is defined as a method for the Analog Class and is called by `asiSendControlStmts`. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_session	Specifies the session object.

Value Returned

`t`	Returns `t` when the keep list is sent to Cadence SPICE.
`nil`	Returns `nil` if there is an error.

Example for Integrators

```
defmethod( asiSendKeepList (( session <yourSimulator>_session ))
    <insert any code you need>
)
```

asiSendModelPath

```
asiSendModelPath( o_session )
     => t | nil
```

Description

Sends the model path to Cadence SPICE.

Note for Integrators: This function is defined as a method for the Analog Class. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_session	Specifies the session object.

Value Returned

t	Returns t when the model path is sent to Cadence SPICE.
nil	Returns nil if there is an error.

Example for Integrators

```
defmethod( asiSendModelPath (( session <yourSimulator>_session ))
    <insert any code you need>
)
```

asiSendNetlist

```
asiSendNetlist( o_session )
     => t | nil
```

Description

Sends the raw netlist to Cadence SPICE using the Cadence SPICE `sim` command.

For more information about the `sim` command, refer to the *Cadence SPICE Reference.*

Note for Integrators: This function is defined as a method for the Analog Class. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

`o_session`	Specifies the session object.

Value Returned

`t`	Returns `t` when the raw netlist is sent to Cadence SPICE.
`nil`	Returns `nil` if there is an error.

Example for Integrators

```
defmethod( asiSendNetlist (( session <yourSimulator>_session ))
    <insert any code you need>
)
```

asiSendNodeSets

```
asiSendNodeSets( o_session )
     => t | nil
```

Description

Places all the nodesets in <*netlistDirectory*>/raw/nodesets and sends a ptprop command to Cadence SPICE.

There is a mechanism in the simdot.f file that gets the value of the ptprop command (if the ptprop command was sent). If the ptprop command was sent, the nodesets file is automatically included in the final netlist. If the ptprop command was not sent, the nodesets file is not included. The nodesets are of the form:

```
.nodeset v(node)=value
```

Note for Integrators: This function is defined as a method for the Analog Class and is called by asiSendControlStmts. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_session	Specifies the session object.
nil	Returns nil if there is an error.

Value Returned

t	Returns t when the nodesets are sent to Cadence SPICE.
nil	Returns nil if there is an error.

Example for Integrators

```
defmethod(asiSendNodeSets(( session <yourSimulator>_session ))
    let( ( nodesetStr )

        nodesetStr = ".nodeset v(%s) = %s\n"
        asiiSendNodeSets( session nodesetStr )

        t
    )
)
```

asiSendOptions

```
asiSendOptions( o_session )
     => t | nil
```

Description

Sends the simulation options to Cadence SPICE.

Note for Integrators: This function is defined as a method for the Analog Class. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_session	Specifies the session object.

Value Returned

t	Returns t when the simulation options are sent to Cadence SPICE.
nil	Returns nil if there is an error.

Example for Integrators

```
defmethod( asiSendOptions ( ( session XYZ_session ))
    let( ( sendMethod simulatorOptionVariableList value command )

        simulatorOptionVariableList = asiGetSimOptionNameList(
            session )

        foreach( name simulatorOptionVariableList

            value = asiGetSimOptionVal( session name )
            sendMethod = asiGetSimOptionSendMethod( session name )

            if( artBlankString( value )  then
                 sprintf( command "deprop XYZ_Opt %s\n" name )
            else
                 caseq( sendMethod
                        ( set
                             sprintf( command "set %s=%s\n"
                                name value )
                        )
                        ( ptprop
                             sprintf( command "ptprop XYZ_Opt
                                %s %s\n"name value )
                        )
```

```
                        ( psprop
                             sprintf( command "psprop XYZ_Opt
                                %s \"%s\" \n"name value )
                        )
                        ( t
                             warn( "don't know how to send XYZ
                                option %s\n." name  )
                             command = nil
                        )
                    )
               )
               when( command
                       asiSendSim( session command nil nil nil )
               )
          )

          t
     )
)
```

asiSendRestore

```
asiSendRestore( o_session )
     => t
```

Description

If DC restore is *on*, send the commands to restore the DC node voltages to Cadence SPICE. If DC restore is *off*, sends the commands to turn off the DC restore function to Cadence SPICE. This function also works for the transient restore function.

Note for Integrators: This routine is defined as a method for the Analog Class. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_session	Specifies the session object.

Value Returned

t	Sends the commands to Cadence SPICE and returns t.

Example for Integrators

```
defmethod( asiSendRestore ( ( session <yourSimulator>_session ) )
    <insert any code you need>
)
```

asiSendUpdateFile

```
asiSendUpdateFile( o_session )
     => t | nil
```

Description

Sends the `update.s` file to Cadence SPICE.

Note for Integrators: This function is defined as a method for the Analog Class. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_session	Specifies the session object.

Value Returned

`t`	Returns `t` when the update file is sent to Cadence SPICE.
`nil`	Returns `nil` if there is an error.

Example for Integrators

```
defmethod( asiSendUpdateFile (( session <yourSimulator>_session ))
    <insert any code you need>
)
```

12

Keep Option Functions

The keep option functions let you specify whether or not *all* the outputs of a particular type of signal are saved during simulation. For example, you might use a keep option function to save all the node voltages or all the port currents.

For information about functions that let you save specific signals, refer to Chapter 12, "Miscellaneous Functions."

asiAddKeepOption

```
asiAddKeepOption(
     o_tool
     ?name          s_name
     [?prompt       t_prompt]
     [?type         s_type]
     [?choices      l_choices]
     [?itemsPerRow  x_itemsPerRow]
     [?value        g_value]
     [?min          g_min]
     [?max          g_max]
     [?allowExpr    s_allowExpr]
     [?row          x_row]
     [?column       x_column]
     [?width        x_width]
     [?coordinates  l_coordinates]
     [?displayOrder x_displayOrder]
     [?labelText    t_labelText]
     [?private      s_private]
     [?display      s_display]
     [?editable     s_editable]
     [?appCB        s_appCB]
     [?callback     t_callback]
     [?formApplyCB  s_formApplyCB]
     [?changeCB     st_changeCB]
     [?doubleClickCBst_doubleClickCB]
     [?numRows      x_numRows]
     [?multipleSelects_multipleSelect]
     [?invalidateFuncs_invalidateFunc]
     )
     => o_envVar | nil
```

Description

Adds a simulator keep option variable.

Arguments

o_tool	Simulation tool object.
s_name	Name of the keep option.
t_prompt	Optional argument that specifies the prompt (on the UI form) for the given option. Default Value: *s_name*

s_type
Type of the option.
Valid Values: `string, integer, float, toggle, cyclic, radio, boolean, list, radioToggle, listBox, fileNam`e (string type for file names only), `label` (for the label on the UI form), `frame` (for a graphic frame around a field), `separator` (for the separator line on the UI form), `button` (for a button on the UI form), `scale` (for a slider field on the UI form)
Default Value: `string`

Note: See the asiAddSimOption function in the "Simulator Option Functions" chapter for an example that shows how to use the *separator* argument.

l_choices
List of choices if *s_type* is `cyclic, radio, radioToggle` or `listBox`, or the list of switches if *s_type* is `toggle`. This argument is valid only if *s_type* is `cyclic`, `toggle, radio, radioToggle`, or `listBox`.

x_itemsPerRow
Numbers of choices per row for `radio`, `cyclic`, `toggle`, and `radioToggle` fields.
Default Value: Total number of choices specified in *l_choices*

g_value
Default value of the option.

g_min
Specifies the minimum value of an `integer`, `float`, or `scale` option.
Default Value: `nil`, which means `-infinity`

g_max
Specifies the maximum value of an `integer, float,` or `scale` option.
Default Value: `nil`, which means `+infinity`

s_allowExpr
Specifies whether *g_value* can contain expressions.
Valid Values: `t` (value can contain expressions), `nil` (value cannot contain expressions)
Default Value: `nil`

x_row
Row in the form where the field appears. This argument is valid only for *'twoD* type forms.

x_column
Column in the form where the field appears. The fields are created according to their required widths and are not meant to

	align with fields in other rows of the form. This argument is valid only for *'twoD* type forms.
x_width	Specifies the width of the field in relation to other fields on the form. Numbers that you enter for this argument are relative values whose actual values are determined by the amount of space available. For example, assume there are three fields for a row that is 400 pixels wide. If the first two fields have an *x_width* of 1, and the last field has an *x_width* of 2, then the widths for the fields are as follows: 100, 100, and 200 pixels. This argument is valid only for *'twoD* type forms. Default Value: 1
l_coordinates	List specifying the coordinates for the field on the UI form. (The format is the same as for the corresponding *hi* field.) This argument is valid only for *'custom* type forms.
x_displayOrder	Position (from the top) of the option in the form. Use *x_displayOrder* to reposition your options in an *inherited* form. By default, the options appear in the order defined; therefore, inherited options appear first. The value of *x_displayOrder* must be an integer that specifies the position you want relative to the top of the form. If more than one option has the same display order integer, the last option found (yours) takes precedence. The remaining options shift down on the form. This argument is valid only for *'oneD* type forms. Valid Values: Any integer
t_labelText	Optional label displayed with frame type fields. Default Value: `nil`
s_private	Optional argument that declares an option as private, which means the option is not visible to the user. Private options are not part of the UI and their values are not saved when a user saves the environment. You might use this argument for values that are constant for all users of the software. Valid Values: `t` (option does not appear in the UI), (option appears in the UI) Default Value: `nil`

s_display Specifies an expression that determines whether the field is to be displayed on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

s_editable Specifies an expression that determines whether the field is to be editable on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

Note: This argument only applies to type-in fields.

s_appCB Specifies a callback function that is executed when the value of the option is changed.
Callback parameter list: (*o_session*)

t_callback Specifies a callback function that is executed when a field is changed on the form while the form is displayed. (You can use this function to do error checking on the value entered by the designer.)

s_formApplyCB Specifies a callback function that is executed when the designer clicks on *Apply* or *OK* on the form containing the associated field.
Callback parameter list: (*o_session r_form r_field*)

st_changeCB Specifies a callback function that is executed when the value of a `listBox` type field is changed on the form.

st_doubleClickCB Specifies a callback function that is executed when a designer double clicks on a `listBox` type field.

x_numRows Number of rows shown on the form for a `listBox` type field.

s_multipleSelect Boolean flag that specifies whether multiple items can be selected from the `listBox` type field.
Valid Values: `t` (multiple items can be selected), `nil` (only one item can be selected at a time)

s_invalidateFunc

Specifies a function that is executed when the value of the option is changed. This function invalidates a step in the flowchart. Callback parameter list: (*o_session*)

Value Returned

o_envVar	Returns the keep option environment variable object created by the procedure.
nil	Returns nil if there is an error.

Example

```
asiAddKeepOption( tool
    ?name           'allDigitalNV
    ?prompt         "Select all digital node voltages"
    ?type           'boolean
    ?value          t
)
```

Adds a keep option to the tool to save all digital node voltages.

Related Functions

To display the current set of keep options, see the asiDisplayKeepOption function on page 12-475.

To change the display characteristics of your Keep Options form, see the asiChangeKeepOptionFormProperties function on page 12-472.

asiChangeKeepOption

```
asiChangeKeepOption(
     o_tool
     ?name          s_name
     [?prompt       t_prompt]
     [?type         s_type]
     [?choices      l_choices]
     [?itemsPerRow  x_itemsPerRow]
     [?value        g_value]
     [?min          g_min]
     [?max          g_max]
     [?allowExpr    s_allowExpr]
     [?row          x_row]
     [?column       x_column]
     [?width        x_width]
     [?coordinates  l_coordinates]
     [?displayOrder x_displayOrder]
     [?labelText    t_labelText]
     [?private      s_private]
     [?display      s_display]
     [?editable     s_editable]
     [?appCB        s_appCB]
     [?callback     t_callback]
     [?formApplyCB  s_formApplyCB]
     [?changeCB     st_changeCB]
     [?doubleClickCBst_doubleClickCB]
     [?numRows      x_numRows]
     [?multipleSelects_multipleSelect]
     [?invalidateFuncs_invalidateFunc]
     )
     => o_envVar | nil
```

Description

Modifies an existing keep option variable for a simulator.

Arguments

o_tool	Simulation tool object.
s_name	Name of the keep option.
t_prompt	Optional argument that specifies the prompt (on the UI form) for the given option. Default Value: *s_name*

s_type
Type of the option.
Valid Values: `string, integer, float, toggle, cyclic, radio, boolean, list, radioToggle, listBox, fileName` (string type for file names only), `label` (for the label on the UI form), `frame` (for a graphic frame around a field), `separator` (for the separator line on the UI form), `button` (for a button on the UI form), `scale` (for a slider field on the UI form)
Default Value: `string`

Note: See the asiAddSimOption function in the "Simulator Option Functions" chapter for an example that shows how to use the *separator* argument.

l_choices
List of choices if *s_type* is `cyclic, radio, radioToggle` or `listBox`, or the list of switches if *s_type* is `toggle`. This argument is valid only if *s_type* is `cyclic,toggle,radio,radioToggle`, or `listBox`.

x_itemsPerRow
Numbers of choices per row for `radio, cyclic, toggle`, and `radioToggle` fields.
Default Value: Total number of choices specified in *l_choices*

g_value
Default value of the option.

g_min
Specifies the minimum value of an `integer`, `float`, or `scale` option.
Default Value: `nil`, which means `-infinity`

g_max
Specifies the maximum value of an `integer`, `float`, or `scale` option.
Default Value: `nil`, which means `+infinity`

s_allowExpr
Specifies whether *g_value* can contain expressions.
Valid Values: `t` (value can contain expressions), `nil` (value cannot contain expressions)
Default Value: `nil`

x_row
Row in the form where the field appears. This argument is valid only for *'twoD* type forms.

x_column
Column in the form where the field appears. The fields are created according to their required widths and are not meant to

	align with fields in other rows of the form. This argument is valid only for *'twoD* type forms.
x_width	Specifies the width of the field in relation to other fields on the form. Numbers that you enter for this argument are relative values whose actual values are determined by the amount of space available. For example, assume there are three fields for a row that is 400 pixels wide. If the first two fields have an *x_width* of 1, and the last field has an *x_width* of 2, then the widths for the fields are as follows: 100, 100, and 200 pixels. This argument is valid only for *'twoD* type forms. Default Value: 1
l_coordinates	List specifying the coordinates for the field on the UI form. (The format is the same as for the corresponding *hi* field.) This argument is valid only for *'custom* type forms.
x_displayOrder	Position (from the top) of the option in the form. Use *x_displayOrder* to reposition your options in an *inherited* form. By default, the options appear in the order defined; therefore, inherited options appear first. The value of *x_displayOrder* must be an integer that specifies the position you want relative to the top of the form. If more than one option has the same display order integer, the last option found (yours) takes precedence. The remaining options shift down on the form. This argument is valid only for *'oneD* type forms. Valid Values: Any integer
t_labelText	Optional label displayed with frame type fields. Default Value: `nil`
s_private	Optional argument that declares an option as private, which means the option is not visible to the user. Private options are not part of the UI and their values are not saved when a user saves the environment. You might use this argument for values that are constant for all users of the software. Valid Values: `t` (option does not appear in the UI), `nil` (option appears in the UI) Default Value: `nil`

s_display Specifies an expression that determines whether the field is to be displayed on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

s_editable Specifies an expression that determines whether the field is to be editable on the UI form. This expression is evaluated when the form is first displayed and whenever a callback is executed on *any* field in the form.
Default Value: `t`

Note: This argument only applies to type-in fields.

s_appCB Specifies a callback function that is executed when the value of the option is changed.
Callback parameter list: (*o_session*)

t_callback Specifies a callback function that is executed when a field is changed on the form while the form is displayed. (You can use this function to do error checking on the value entered by the designer.)

s_formApplyCB Specifies a callback function that is executed when the designer clicks on *Apply* or *OK* on the form containing the associated field.
Callback parameter list: (*o_session r_form r_field*)

st_changeCB Specifies a callback function that is executed when the value of a `listBox` type field is changed on the form.

st_doubleClickCB Specifies a callback function that is executed when a designer double clicks on a `listBox` type field.

x_numRows Number of rows shown on the form for a `listBox` type field.

s_multipleSelect Boolean flag that specifies whether multiple items can be selected from the `listBox` type field.
Valid Values: `t` (multiple items can be selected), `nil` (only one item can be selected at a time)

s_invalidateFunc

Specifies a function that is executed when the value of the option is changed. This function invalidates a step in the flowchart. Callback parameter list: (*o_session*)

Value Returned

o_envVar	Returns the new keep option object.
nil	Returns nil if there is an error.

Example

```
asiChangeKeepOption( asiGetTool('XYZ)
    ?name       'allAnalogTC
    ?value      t
)
```

Changes the default value of the allAnalogTC keep option to t.

Related Functions

To display the current set of keep options, see the asiDisplayKeepOption function on page 12-475.

To change the display characteristics of your Keep Options form, see the asiChangeKeepOptionFormProperties function on page 12-472.

asiChangeKeepOptionFormProperties

```
asiChangeKeepOptionFormProperties(
     o_tool
     [?types_type]
     [?widthx_width]
     [?columnsx_columns]
     )
     => o_formObj | nil
```

Description

Changes the display characteristics of the Keep Options form.

Arguments

o_tool	Simulation tool object.
s_type	Specifies the form type. Valid Values:

'oneD	Specifies a sequential display of fields in one column.
'twoD	Specifies a two dimensional display of fields based on row and column positions. (Use the asiAddKeepOption or asiChangeKeepOption function to specify values for the rows and columns.)
'custom	Lets you specify the exact coordinate locations for each field. (Use the asiAddKeepOption or asiChangeKeepOption function to specify the coordinates.)
'matrix	Specifies a matrix of equally sized fields.

Default Value: 'oneD

x_width	Width of the form, in pixels. Default Value: The minimum default width of the form is 400 pixels. If the fields require more space than this, the form defaults to the smallest width that can accommodate the fields.

x_columns	Number of columns. Use this argument only with matrix type forms. Default Value: 2

Value Returned

o_formObj	Returns the keep option form object if successful.
nil	Returns nil otherwise.

Example

```
asiChangetKeepOptionFormProperties( asiGetTool('spectreS)
?width 300 )
```

For the Spectre simulator, changes the width of the Keep Options form to 300.

asiDeleteKeepOption

```
asiDeleteKeepOption( o_tool s_name )
     => t | nil
```

Description

Deletes a simulator keep option variable.

Arguments

o_tool	Simulation tool object.
s_name	Name of the keep option variable to delete.

Value Returned

`t`	Returns `t` when the keep option variable is deleted.
`nil`	Returns `nil` if the option does not exist.

Example

```
asiDeleteKeepOption( asiGetTool('XYZ)
    'allAnalogTC
)
```

Deletes the keep option to save all analog terminal currents for the XYZ simulator.

asiDisplayKeepOption

```
asiDisplayKeepOption( o_tool )
     => t | nil
```

Description

Displays the current simulator keep option names. Use this function to determine which options you want to modify. Do not use this function as part of another procedure.

Arguments

o_tool	Simulation tool object.

Value Returned

`t`	Displays the current keep option names and returns `t`.
`nil`	Returns `nil` if there is an error.

Example

```
asiDisplayKeepOption( asiGetTool( 'spectreS ))
```

Displays the keep option names for the *spectreS* simulator.

asiDisplayKeepOptionFormProperties

```
asiDisplayKeepOptionFormProperties( o_tool )
     => t | nil
```

Description

Displays the form characteristics for the Keep Options form. Use this function only to determine which form characteristics you want to modify. Do not use this function as part of another procedure.

Arguments

o_tool	Simulation tool object.

Value Returned

`t`	Displays a list of the Keep Option form characteristics and returns `t`.
`nil`	Returns `nil` otherwise.

Example

```
asiDisplayKeepOptionFormProperties( asiGetTool('spectreS))
```

For the Spectre simulator, displays the form characteristics for the Keep Options form and returns `t`.

asiGetKeepOptionChoices

```
asiGetKeepOptionChoices(
     {o_session | o_tool}
     s_name
     )
     => l_choices | nil
```

Description

Gets the list of choices for a keep option that is set up as a list box.

Arguments

o_session	Simulation session object.
o_tool	Simulation tool object.
s_name	Name of the option for which you want the list of choices.

Value Returned

l_choices	Returns the list of choices for the specified keep option.
nil	Returns nil if the option does not exist.

Related Function

To display the current set of keep options, see the asiDisplayKeepOption function on page 12-475.

asiGetKeepOptionVal

```
asiGetKeepOptionVal(
     { o_tool | o_session }
     s_name
     )
     => g_value
```

Description

Gets the value of a keep option variable for a tool or session object.

Arguments

o_tool	Simulation tool object.
o_session	Simulation session object.
s_name	Name of the simulation option variable for which you want to get the value.

Value Returned

g_value	Returns the value for the keep option variable.

Example

```
asiGetKeepOptionVal( asiGetTool( 'XYZ ) 'allAnalogTC )
```

Gets the value of the `allAnalogTC` keep option for the XYZ simulator.

asiInit<yourSimulator>KeepOption

```
asiInit<yourSimulator>KeepOption( o_tool )
     => t
```

Description

Calls the procedures to initialize your simulator keep option variables.

Note: You must write `asiInit<`*`yourSimulator`*`>KeepOption`, where *<yourSimulator>* is the name of your simulator. Do not include the angle brackets (<>).

Arguments

o_tool	Simulation tool object.

Value Returned

`t`	Returns `t` when your procedures for initializing the keep option variables are called.

Note: Write `asiInit<`*`yourSimulator`*`>KeepOption` to return `t`.

Example

```
procedure( asiInitXYZKeepOption( tool )
    <insert your code>
    t
)
```

Creates the procedure that initializes the keep options for the XYZ simulator.

asiSetKeepOptionChoices

```
asiSetKeepOptionChoices(
     {o_session | o_tool}
     s_name
     l_choices
     )
     => l_choices | nil
```

Description

Specifies the list of choices to appear in the list box field for the specified keep option.

Arguments

o_session	Simulation session object.
o_tool	Simulation tool object.
s_name	Name a keep option of the type `listBox`.
l_choices	List of choices to appear in the list box field.

Value Returned

l_choices	Returns the new list of choices to appear in the list box field.
`nil`	Returns `nil` if the option does not exist.

asiSetKeepOptionVal

```
asiSetKeepOptionVal(
     { o_tool | o_session }
     s_name
     g_value
     )
     => g_value | nil
```

Description

Sets the value for the specified keep option variable for a tool or session object.

Arguments

o_tool	Simulation tool object.
o_session	Simulation session object.
s_name	Name of the keep option variable that you want to set.
g_value	Value for the keep option variable.

Value Returned

g_value	Returns the new value for the keep option.
nil	Returns nil if the keep option does not exist.

Example

```
asiSetKeepOptionVal( session 'allAnalogNV t)
```

Saves all analog node voltages for the specified session.

13

Direct Plot Functions

The direct plot functions calculate the simulation result of some pre-defined analysis. Currently, direct plot SKILL functions are available to calculate simulation results for RF circuits only.

ADE L provides the following direct plot SKILL functions:

Function	Use
drplACPRWithMask	Plots `acpr` and `spectrum` masks if one of standards defined in Channel Definitions on `envlp` result Direct Plot form is selected.
drplPacVolGnExpDen	Plots the PAC voltage wave.
drplJitter	Calculates jitter for synchronous or autonomous circuits.
drplRFJc	Calculates cycle jitter for single event.
drplRFJcc	Calculates cycle-to-cycle jitter for single event.
drplParamSweepRFJc	Calculates cycle jitter for parametric sweep with multiple events.
drplParamSweepRFJcc	Calculates cycle-to-cycle jitter for parametric sweep with multiple events.
drplSwpHp	Returns the hybrid matrix for sweep port SP analysis.
drplSwpSp	Returns the S-parameter waveform for sweep port SP analysis.
drplSwpYp	Returns the admittance matrix for sweep port SP analysis.
drplSwpZm	Returns the port input impedance matrix for sweep port SP analysis.

drplSwpZp	Returns the impedance matrix for sweep port SP analysis.

drplACPRWithMask

```
drplACPRWithMask(
     acprw
     sig
     )
     => waveform | nil
```

Description

This function plots `acpr` and `spectrum` masks if you select one of standards defined in Channel Definitions on `envlp` result Direct Plot form. These masks confirm that the `acpr` reaches the mask requirement defined in communication standards.

Arguments

acprw	Name of the `acpr` wave.
sig	The signal standard, such as 802_11a.

Value Returned

waveform	Returns the waveform of `acpr` and `mask`.
`nil`	Returns `nil`, if result `envlp_fd` does not exist or power spectral density can not be calculated.

Example

```
drplACPRWithMask(db10(psdbb(real(harmonic(v("/net9" ?result "envlp_fd") '1))
imag(harmonic(v("/net9" ?result "envlp_fd") '1)) 0 0.00016 12800 ?windowSize 64
?windowName "Cosine4" ?detrending "none")) "802_11a")
```

The above example returns a waveform that shows sweep variable on the x-axis and jitter value plotted on the y-axis.

drplEvmBpsk

```
drplEvmBpsk(
     o_waveform1
     o_waveform2
     n_tDelay
     n_sampling
     b_autoLevelDetect
     n_voltage
     n_offset
     b_normalize
     b_percent
     )
     => o_waveform | nil
```

Description

Processes the I and Q waveform outputs from the transient simulation run to calculate the Error Vector Magnitude (EVM) and plot the I versus Q scatterplot. EVM is a useful measurement to describe the overall signal amplitude and phase modulated signal quality. It is based on a statistical error distribution normalized from an ideal digital modulation. Binary Phase Shift Keying (BPSK) is a typical modulation scheme where EVM is useful. The EVM is calculated by detecting the I and Q signal levels corresponding to the two possible I and Q symbol combinations, and calculating the difference between the actual signal level and the ideal signal level. Compared to other types of phase shift keying, such as QPSK, QAM16 and QAM64, BPSK has lowest bit error rate for the same signal to noise ratio.

Note: This function is not supported for family of waveforms.

Arguments

o_waveform1	The waveform for I signal.
o_waveform2	The waveform for Q signal.
n_tDelay	The start time for the first valid symbol. This can be obtained from the *Waveform Viewer* window by recording the time of the first minimum or maximum of the valid symbol (whichever is earlier), on the selected signal stream.
n_sampling	A period for the symbol. Each period is represented by a data rate. The data rate at the output is determined by the particular modulation scheme being used.

b_autoLevelDetect	Indicates that you want the amplitude, `n_voltage`, and DC offset, `n_offset`, to be calculated automatically. Amplitude is calculated by averaging the rectified voltage level of the signal streams. DC offset is calculated by averaging the sum of an equal number of positive and negative symbols in each signal stream. These values are used to determine the EVM value. If the `b_autoLevelDetect` is `nil`, the values for `n_voltage` and `n_offset` are required. Valid values: `t`, `nil` Default value: `t`
n_voltage	The amplitude of the signal.
n_offset	The DC offset value.
b_normalize	An option to see the scatter plot normalized to the ideal values `+1` and `-1` (for example, when superimposing scatter plots from different stages in the signal flow, where the levels may be different, but you want to see relative degradation or improvement in the scatter). This option does not affect the calculation of the EVM number. Valid values: `t`, `nil` Default value: `nil`
b_percentage	Specifies whether to print the average EVM in percentage or in dB scale. If the value is set to `t`, average EVM is printed in percentage, otherwise, average EVM is printed in dB scale. Valid values: `t`, `nil` Default value: `nil`

Value Returned

`o_waveform`	Returns a waveform object representing the EVM value computed from the waveforms.
`nil`	Returns `nil` and prints the error message if the function is unsuccessful.

Examples

Example 1

```
drplEvmBpsk( v("samp_out_Q"), v("samp_out_I") 1.5u, 181.81n, t, nil, nil, nil)
```

Calculates the EVM value when `b_autoLevelDetect` is set to `t` . In this case, values are not specified for `n_voltage` and `n_offset`.

Example 2

```
drplEvmBpsk( v("samp_out_Q"), v("samp_out_I") 1.5u, 181.81n, nil, 1.3, 0, nil)
```

Calculates the EVM value when `b_autoLevelDetect` is set to `nil`. In this case, values are specified for `n_voltage` and `n_offset`.

drplPacVolGnExpDen

```
drplPacVolGnExpDen(
     denSigStr
     rh
     name
     )
     => waveform | nil
```

Description

This function plots the PAC voltage wave.

Arguments

denSigStr	The defined signal string.
rh	The reference harmonic.
name	Name.

Value Returned

waveform	Returns the waveform of PAC voltage.
nil	Returns nil, otherwise.

Example

```
drplPacVolGnExpDen("v(\"/VLO\" ?result \"hbac\")" '(0) nil)
```

drplJitter

```
drplJitter(
     result
     resultsDir
     freq
     { k | 1 }
     unit
     ber
     event
     )
     => value| nil
```

Description

Calculates FM jitter at the output of an autonomous circuit or PM jitter at the output of a driven circuit. This function uses the results of the pnoise analysis. The type of the circuit as autonomous or driven is determined from the PSS result.

Arguments

result	Name of the result of pnoise analysis. By default, the name for the result of pnoise jitter analysis is `pnoise_pmjitter`.
resultsDir	Path to the results directory.
freq	Frequency at which you want to calculate jitter. This value is used in case of an autonomous circuit only. Specify it as `nil`, in case of a driven circuit.
k	The number of cycles. The default value of k is `1`.
unit	Unit to measure jitters. Possible values are `Second`, `UI`, and `ppm`.
ber	The value of BER (Bit Error Rate) when the signal level is peak-to-peak.
event	Index of an event in the result. The jitter value is calculated for each event.

Value Returned

value	Returns the jitter value.
nil	Returns nil, if the result name is incorrect and jitter cannot be calculated.

Example

```
drplJitter(?result "pnoise_pmjitter" ?unit "Second" ?event 3 ?k 1)
```

The above SKILL function returns a jitter value.

drplRFJc

```
drplRFJc(
     from
     to
     k
     multiplier
     result
     resultsDir
     unit
     ber
     eventList
     )
     => value | nil
```

Description

Calculates cycle jitter from the result for a single event of pnoise analysis.

Arguments

from	The lower frequency limiter.
to	The upper frequency limiter.
k	The number of cycles.
multiplier	The frequency multiplier. By default, it is set to `1`.
result	Name of the result of pnoise jitter analysis.
resultsDir	Path to the results directory.
unit	Unit to measure jitters. Possible values are `Second`, `UI`, and `ppm`.
ber	The value of BER (Bit Error Rate) when the signal level is peak-to-peak.
event	Index ID of an event.

Value Returned

value	Returns the value of cycle jitter.
nil	Returns nil, if the result name is incorrect and jitter cannot be calculated.

Example

```
drplRFJc( ?from 100 ?to 50000000 ?k 1 ?result "pnoise_pmjitter" ?unit "Second"
?event 0 )
```

Returns the cycle jitter value.

drplRFJcc

```
drplRFJcc(
     from
     to
     k
     multiplier
     result
     resultsDir
     unit
     ber
     event
     )
     => value | nil
```

Description

Calculates cycle-to-cycle jitter from the result of pnoise analysis for a single event.

Arguments

from	The lower frequency limiter.
to	The upper frequency limiter.
k	The number of cycles.
multiplier	The frequency multiplier. By default, it is set to `1`.
result	Name of the result of pnoise jitter analysis.
resultsDir	Path to the results directory.
unit	Unit to measure jitters. Possible values are `Second`, `UI`, and `ppm`.
ber	The value of BER (Bit Error Rate) when the signal level is peak-to-peak.
event	Index of an event in the result.

Value Returned

value	Returns the value of cycle jitter.
nil	Returns nil, if the result name is incorrect and jitter cannot be calculated.

Example

```
drplRFJcc( ?from 100 ?to 50000000 ?k 1 ?result "pnoise_pmjitter" ?unit "Second"
?event 0)
```

Returns cycle-to-cycle jitter value.

drplParamSweepRFJc

```
drplParamSweepRFJc(
     from
     to
     k
     multiplier
     result
     resultsDir
     unit
     ber
     eventList
     )
     => waveform | nil
```

Description

Calculates cycle jitter for parametric sweep with multiple events. This function calculates jitter using the Pnoise result for driven circuits only.

Arguments

from	The lower frequency limiter.
to	The upper frequency limiter.
k	The number of cycles.
multiplier	The frequency multiplier. By default, it is set to `1`.
result	Name of the result of pnoise analysis.
resultsDir	Path to the results directory.
unit	Unit to measure jitters. Possible values are `Second`, `UI`, `ppm`.
ber	The value of BER (Bit Error Rate) when the signal level is peak-to-peak.
eventList	List of index ID for each event of parameter sweep.

Value Returned

waveform	Returns the waveform of cycle jitter.
nil	Returns nil, if the result name is incorrect and jitter cannot be calculated.

Example

```
drplParamSweepRFJc (?from 1000000 ?to 100000000 ?k 1 ?multiplier 1 ?result
"pnoise_pmjitter" ?unit "Second" ?eventList '("-10.0   (1  1.25143e-10)" "-10.0
(2  6.57854e-10)" "-9.0   (1  1.27759e-10)" "-9.0   (2  6.5234e-10)"))
```

The above example returns a waveform that shows sweep variable on the x-axis and jitter value plotted on the y-axis.

drplParamSweepRFJcc

```
drplParamSweepRFJcc(
     from
     to
     k
     multiplier
     result
     resultsDir
     unit
     ber
     eventList
     )
     => waveform | nil
```

Description

Calculates cycle jitter for parametric sweep with multiple events. This function calculates jitter using the Pnoise result for driven circuits only.

Arguments

from	The lower frequency limiter.
to	The upper frequency limiter.
k	The number of cycles.
multiplier	The frequency multiplier. By default, it is set to `1`.
result	Name of the result of pnoise analysis.
resultsDir	Path to the results directory.
unit	Unit to measure jitters. Possible values are `Second`, `UI`, and `ppm`.
ber	The value of BER (Bit Error Rate) when the signal level is peak-to-peak.
event	List of index ID for each event of parameter sweep.

Value Returned

waveform	Returns the waveform of cycle jitter.
nil	Returns nil, if the result name is incorrect and jitter cannot be calculated.

Example

```
drplParamSweepRFJc ?from 1000000 ?to 100000000 ?k 1 ?multiplier 1 ?result
"pnoise_pmjitter" ?unit "Second" ?eventList '("-10.0   (1  1.25143e-10)" "-10.0
(2  6.57854e-10)" "-9.0   (1  1.27759e-10)" "-9.0   (2  6.5234e-10)")
```

Returns cycle-to-cycle jitter value and its corresponding waveform.

The above example returns a waveform that shows sweep variable on the x-axis and jitter value is plotted on the y-axis.

drplSwpHp

```
drplSwpHp(
     iIndex
     jIndex
     result
     resultsDir
     )
     => waveform | nil
```

Description

Returns the hybrid matrix for sweep port SP analysis.

Arguments

iIndex	Index of the first port.
jIndex	Index of the second port.
result	Name of the result of SP analysis.
resultsDir	Path to the results directory.

Value Returned

waveform	Returns the waveform of cycle jitter.
nil	Returns nil, if the result name is incorrect and jitter cannot be calculated.

Example

```
drplSwpHp( 1 1 ?result "sp" )
```

Returns a waveform with frequency on the x-axis and the hybrid matrix on the y-axis.

drplSwpSp

```
drplSwpSp(
     iIndex
     jIndex
     result
     resultsDir
     )
     => waveform | nil
```

Description

Returns the S-parameter waveform for sweep port SP analysis.

Arguments

iIndex	Index of the first port.
jIndex	Index of the second port.
result	Name of the result of SP analysis.
resultsDir	Path to the results directory.

Value Returned

waveform	Returns the waveform of cycle jitter.
nil	Returns nil.

Example

```
drplSwpSp( 1 1 ?result "sp" )
```

Returns a waveform with frequency on the x-axis and the S-parameter waveform on the y-axis.

drplSwpYp

```
drplSwpYp(
     iIndex
     jIndex
     result
     resultsDir
     )
     => waveform | nil
```

Description

Returns the admittance matrix for sweep port SP analysis.

Arguments

iIndex	Index of the first port.
jIndex	Index of the second port.
result	Name of the result of SP analysis.
resultsDir	Path to the results directory.

Value Returned

waveform	Returns the waveform of cycle jitter.
nil	Returns nil.

Example

```
drplSwpYp( 1 2 ?result "yp" )
```

Returns a waveform with frequency on the x-axis and the admittance matrix on the y-axis.

drplSwpZm

```
drplSwpZm(
     iIndex
     jIndex
     result
     resultsDir
     )
     => waveform | nil
```

Description

Returns the port input impedance waveform for sweep port SP analysis.

Arguments

iIndex	Index of the first port.
jIndex	Index of the second port.
result	Name of the result of SP analysis.
resultsDir	Path to the results directory.

Value Returned

waveform	Returns the waveform of cycle jitter.
nil	Returns nil.

Example

```
drplSwpZm( 2 2 ?result "sp")
```

Returns a waveform with frequency on the x-axis and the port input impedance matrix on the y-axis.

drplSwpZp

```
drplSwpZp(
     iIndex
     jIndex
     result
     resultsDir
     )
     => waveform | nil
```

Description

Returns the impedance matrix for sweep port SP analysis.

Arguments

iIndex	Index of the first port.
jIndex	Index of the second port.
result	Name of the result of SP analysis.
resultsDir	Path to the results directory.

Value Returned

waveform	Returns the waveform of cycle jitter.
nil	Returns nil.

Example

```
drplSwpZp(2 2 ?result "sp")
```

Returns a waveform with frequency on the x-axis and the impedance matrix on the y-axis.

.

14

Data Access Functions

This chapter describes the functions that let you define data access functions and data mapping functions.

For more information, see Chapter 16, “Miscellaneous Functions,”.

asiDefineDataAccessFunction

```
asiDefineDataAccessFunction( o_tool s_dataType s_dataFunction )
     => s_dataFunction
```

Description

Redefines a data access function.

Note: Typically you do not need to redefine a data access function.

Arguments

o_tool	Simulation tool object.
s_dataType	Data type for the data access function. Valid Values: `VT, VF, VS, IT, IF, IS, VDC, IDC, VTRE, OP, OPT, MP, VN2, VNP`, or `VNPP`
s_dataFunction	Name of the data access function that you write to access the data from the Parameter Storage Format (PSF) file.

Value Returned

s_dataFunction	Returns the name of the data access function.

Example

```
asiDefineDataAccessFunction( tool 'VT 'XYZVT )
```

Calls the XYZVT function, which gets the VT information from the PSF file.

```
    procedure( XYZVT(specifier dataDir simData)
        let((wave daf)
            daf = asiGetDataAccessFunction(asiGetTool('analog) 'VT)
            wave = apply( daf list(specifier dataDir simData))
            wave
        )
    )
```

You can use an alternative to `asiDefineDataAccessFunction`, to achieve the same functionality. This would be helpful for user defined data access types. For example, the

existing code would work with VT, IT etc., but the code below would work with a new analysis called "sp".

```
procedure( XYZDataAccessSp(specifier dataDir simData)
    let((wave)
        wave = asiGetDrlData("sp" specifier dataDir)
        wave
    )
)
procedure( XYZMapNetNameSp( dataDir specifier )
    let(()
    specifier = parseString((concat "s" car(specifier) "_" cadr(specifier)))
    specifier
    )
)
```

asiDefineDataMappingFunction

```
asiDefineDataMappingFunction( o_tool s_dataType s_function )
     => s_function
```

Description

Defines the data mapping functions.

Arguments

o_tool	Simulation tool object.
s_dataType	Datatype for the data access function. Valid Values: *VT, VF, VS, IT, IF, IS, VDC, IDC, VTRE, OP, OPT, MP, VN2, VNP*, or *VNPP*

s_function	Name of the data mapping function that you write to convert the Cadence SPICE PSF name to your name. Typically, you use one of the following
	`asi<`*yourSimulator*`>MapNetName`
	`asi<`*yourSimulator*`>MapTerminalName`
	`asi<`*yourSimulator*`>MapInstanceName`
	Callback parameter list: (*s_dataDir l_specifier*)
	s_dataDir specifies the data directory, and *l_specifier* is a list containing the name of the item to look for in the `PSF` file.

Value Returned

s_function	Returns the name of the data mapping function.

Example

```
asiDefineDataMappingFunction( tool 'VT 'asiXYZMapNetName )

procedure( asiXYZMapNetName( dataDir specifier )
    ; replace ^ by . in net names...
    specifier = asiMapNetName( dataDir specifier )
    rplaca(
        specifier
        artStrSubstitute( car( specifier ) "^" "." )
    )
)
```

Calls the `XYZMapNetName` function to convert the *cdsSpice*-like net name to the net name in the PSF file of the XYZ simulator. For example, you might use this function to convert lower case characters to uppercase, or you might convert the hierarchical delimiter (^) to the hierarchical delimiter for your simulator.

asiGetCalcResultsDir

```
asiGetCalcResultsDir()
```

Description

Returns the results directory currently used by calculator functions.

Arguments

None.

Values Returned

t_Datadir	Returns the results directory currently used by calculator functions.
`nil`	Returns `nil` if no results are selected.

Example

```
asiGetCalcResultsDir()
```

Returns the current results directory that is being used by calculator functions.

asiInit<yourSimulator>DataAccessFunction

```
asiInit<yourSimulator>DataAccessFunction( o_tool )
     => t
```

Description

Calls the procedures that modify your data access routines.

Note: You must write `asiInit<yourSimulator>DataAccessFunction`, where *<yourSimulator>* is the name of your simulator. Do not include the angle brackets (<>).

Arguments

o_tool	Simulation tool object.

Value Returned

`t`	Returns `t` when your procedures are called.

Note: You must write this procedure to return `t`.

Example

```
procedure( asiInitXYZDataAccessFunction(tool)
    <insert your code>
    t
)
```

Creates the procedure that calls the functions to modify the data access routines for the XYZ simulator.

All functions described in the following section accept an optional argument called *data-directory*. If this argument is not provided, open results are used by the function. For example, VT("out") will return the transient voltage at the "out" from the currently open results. If the data-directory is provided, this argument will only be used internally and will not alter the currently selected results.

The Ocean commands `openResults` and `selectResult` do not have any effect on these functions. In case multiple analyses of the same type are present, the first found occurrence of that analysis will be used.

VAR

```
VAR(t_variable-name [t_dataDir]) => number | nil
```

Description

Returns the value of the specified design variable.

Arguments

t_variable-name	Name of the variable.
t_dataDir	Directory containing the PSF files(results). When specified, this argument will only be used internally and will not alter the currently selected results.The default is the currently selected results directory.

Value Returned

`number`	Returns the value of the specified design variable.
`nil`	Returns `nil` if the design variable was not found. Prints an error message if the data-directory does not contain valid PSF data.

Examples

```
VAR("R1")
=>  1000.0
VAR("temp" "./simulation/test/spectre/schematic-save")
=>  27.0
```

DATA

```
DATA(t_net_name t_analysis [t_dataDir]) => data | nil
```

Description

This is a basic data access function. It returns data for the specified node and analysis.

Arguments

t_net_name	Net name.
t_analysis	Name of the analysis.
t_dataDir	Directory containing the PSF files (results). When specified, this argument will only be used internally and will not alter the currently selected results.The default is the currently selected results directory.

Value Returned

`data`	The waveform for specified net and analysis.
`nil`	Returns `nil` if no data was found.

Examples

```
DATA("net1" "tran-tran")
=>srrWave:49610776
plot(DATA("net1" "ac-ac"))
=>t (plot window comes up)
```

VS

```
VS(t_name [t_dataDir] ) => wave | nil
```

Description

Returns dc sweep waveform for the specified net.

Arguments

t_net_name	Net name.
t_dataDir	Directory containing the PSF files(results). When specified, this argument will only be used internally and will not alter the currently selected results.The default is the currently selected results directory.

Value Returned

`data`	The dc sweep waveform for specified net.
`nil`	Returns `nil` if no data was found.

Examples

```
plot(VS("out"))
=>t  (plot window comes up)
value(VS("net14") 4)
=> 1.3  (value at sweep variable = 4)
```

OP

```
OP(t_instance_name t_parameter_name [t_dataDir]) => number | nil
```

Description

Returns the value of the operating point parameter for the specified instance.

Arguments

t_instance_name	Instance name.
t_parameter_name	Parameter name.
t_dataDir	Directory containing the PSF files (results). When specified, this argument will only be used internally and will not alter the currently selected results. The default is the currently selected results directory.

Value Returned

`number`	Returns value of the specified operating point.
`nil`	Returns `nil` is the design variable was not found.

Examples

```
OP("R1" "pwr")
=> 0.001
OP("R1" "pwr" "./simulation/test/spectre/schematic-save")
=> 0.002
```

OPT

```
OPT( t_instanceName t_parameter [t_dataDir] )=> data | nil
```

Description

Returns the transient operating point for the specified instance parameter.

Arguments :

t_instance_name	Instance name.
t_parameter_name	Parameter name.
t_dataDir	Directory containing the PSF files(results). When specified, this argument will only be used internally and will not alter the currently selected results.The default is the currently selected results directory.

Value Returned

number	Returns value of the specified operating point
nil	Returns nil is the design variable was not found.

Examples

```
OPT("R7" "pwr")
=>0.0
OPT("R7" "pwr" "./simulation/test/spectre/schematic-save")
=>0.00025
```

MP

```
MP( t_instanceName t_parameter [t_dataDir]) => number | nil
```

Description

Returns the specified model parameter for the instance.

Arguments

t_instance_name	Instance name.
t_parameter_name	Parameter name.
t_dataDir	Directory containing the PSF files(results). When specified, this argument will only be used internally and will not alter the currently selected results. The default is the currently selected results directory.

Value Returned

`number`	Returns value of the specified model parameter.
`nil`	Returns nil is the design variable was not found.

Examples

```
MP("I8.M3" "vpb")
=> -0.13
```

NG

```
NG([t_dataDir]) => nw_noiseGain | nil
```

Description

Returns the noise gain waveform.

Arguments

t_dataDir	Directory containing the PSF files(results). When specified, this argument will only be used internally and will not alter the currently selected results.The default is the currently selected results directory.

Value Returned

`wave`	Returns the noise gain waveform.
`nil`	Returns `nil` if no data was found

Examples

```
NG()
=>srrWave:34428932
NG("./simulation/test/spectre/schematic-save")
=>srrWave:23847792
```

VN

```
VN([t_dataDir]) => nw_noise | nil
```

Description

Returns the noise waveform specified in V/sqrt (Hz).

Arguments

t_dataDir	Directory containing the PSF files(results). When specified, this argument will only be used internally and will not alter the currently selected results.The default is the currently selected results directory.

Value Returned

`wave`	Returns the noise waveform.
`nil`	Returns `nil` if no data was found.

Examples

```
VN()
=>srrWave:82374923
VN("./simulation/test/spectre/schematic-save")
=>srrWave:23749823
```

VN2

```
VN2([t_dataDir)
```

Description

Returns the noise waveform in V^2/Hz.

Arguments

t_dataDir	Directory containing the PSF files(results). When specified, this argument will only be used internally and will not alter the currently selected results. The default is the currently selected results directory.

Value Returned

`wave`	Returns the noise waveform.
`nil`	Returns `nil` if no data was found.

Examples

```
VN2()
=>srrWave:82374923
VN2("./simulation/test/spectre/schematic-save")
=>srrWave:23749823
```

VNP

```
VNP(t_name [t_dataDir])
```

Description

Returns any single level noise parameter available in the PSF database.

Arguments

t_name	Name of the noise parameter.
t_dataDir	Directory containing the PSF files(results). When specified, this argument will only be used internally and will not alter the currently selected results. The default is the currently selected results directory.

Value Returned

`date`	Value of the specified noise parameter.
`nil`	Specified noise parameter not found.

Example

```
VNP("F")
=>srrWave:49610808
plot(VNP("NF"))
=>t  (plot window comes up)
```

VNPP

```
VNPP(t_name t_param [t_dataDir] )
```

Description

VNPP accesses any double level noise parameter available in the PSF database.

Arguments

t_name	Name of the component.
t_param	Name of the noise parameter.
t_dataDir	Directory containing the PSF files(results). When specified, this argument will only be used internally and will not alter the currently selected results.The default is the currently selected results directory.

Value Returned

date	Value of the specified noise parameter.
nil	Specified noise parameter not found.

Example

```
VNPP("R1" "fn")
=>srrWave:49610992
```

VPD

```
VPD( t_net1 t_net2 [t_dataDir] ) => wave | nil
```

Description

Returns the waveform representing phase difference between voltages at the two nets.

Arguments

t_net1	Name of the 1st net.
t_net2	Name of 2nd net.
t_dataDir	Directory containing the PSF files(results). When specified, this argument will only be used internally and will not alter the currently selected results.The default is the currently selected results directory.

Value Returned

`wave`	Phase difference between voltages at the two nets.
`nil`	Returns nil if no data was found.

Examples

```
VPD("net1" "net2")
=>srrWave:49610872
```

VF

```
VF( t_netName [t_dataDir] ) => wave | nil
```

Description

Returns the waveform representing the ac sweep net voltage.

Arguments

t_netName	Name of the net.
t_dataDir	Directory containing the PSF files(results). When specified, this argument will only be used internally and will not alter the currently selected results.The default is the currently selected results directory.

Value Returned

`wave`	Returns the AC sweep voltage for specified net.
`nil`	Returns `nil` if no data was found

Examples

```
VF("out")
=>srrWave:78782738
VF("out" "./simulation/test/spectre/schematic-save")
=>srrWave:43985992
```

IS

```
IS( t_terminal [t_dataDir] )
```

Description

Returns waveform representing the dc sweep terminal current.

Arguments

t_terminal	name of the terminal.
t_dataDir	Directory containing the PSF files(results). When specified, this argument will only be used internally and will not alter the currently selected results.The default is the currently selected results directory.

Value Returned

`data`	Returns the dc sweep current for specified terminal.
`nil`	Returns `nil` if no data was found

Examples

```
IS("/V0/PLUS")
=>srrWave:49090909
IS("/V1/PLUS" "./simulation/test/spectre/schematic-save")
=>srrWave:40349095
```

IT

```
IT( t_terminal [t_dataDir] )
```

Description

Returns waveform representing the transient sweep terminal current.

Arguments

t_terminal	Name of the terminal.
t_dataDir	Directory containing the PSF files(results). When specified, this argument will only be used internally and will not alter the currently selected results.The default is the currently selected results directory.

Value Returned

`data`	Returns the transient current for specified terminal.
`nil`	Returns `nil` if no data was found

Examples

```
IT("/V0/PLUS")
=>srrWave:48729090
IT("/V1/PLUS" "./simulation/test/spectre/schematic-save")
=>srrWave:40345345
```

IF

```
IF( t_terminal [t_dataDir] )
```

Description

Returns the waveform representing the ac sweep terminal current.

Arguments

t_terminal	name of the terminal.
t_dataDir	Directory containing the PSF files(results). When specified, this argument will only be used internally and will not alter the currently selected results.The default is the currently selected results directory.

Value Returned

`data`	Returns the ac current for specified terminal.
`nil`	Returns `nil` if no data was found.

Examples

```
IF("/V0/PLUS")
=>srrWave:49610872
IF("/V1/PLUS" "./simulation/test/spectre/schematic-save")
=>srrWave:44510345
```

IDC

```
IDC(t_terminal [t_dataDir]) => data | nil
```

Description

Returns the waveform representing the DC terminal current.

Arguments

t_terminal	Name of the terminal.
t_dataDir	Directory containing the PSF files(results). When specified, this argument will only be used internally and will not alter the currently selected results.The default is the currently selected results directory.

Value Returned

`data`	Returns the dc current for specified terminal.
`nil`	Returns `nil` if no data was found

Examples

```
IDC("/V0/PLUS")
=>3.143
IDC("/V1/PLUS" "./simulation/test/spectre/schematic-save")
=>4.23
```

VDC

```
VDC(t_netname [t_dataDir]) => data | nil
```

Description

Returns dc voltage for the specified net.

Arguments

t_netname	name of the net.
t_dataDir	Directory containing the PSF files(results). When specified, this argument will only be used internally and will not alter the currently selected results.The default is the currently selected results directory.

Value Returned

`data`	Returns the dc voltage for specified net.
`nil`	Returns `nil` if no data was found

Examples

```
VDC("net15")
=>3.143
VDC("out" "./simulation/test/spectre/schematic-save")
=>4.23
```

SIMULATOR

```
SIMULATOR([t_dataDir]) => name | nil
```

Description

Returns the name of simulator.

Arguments

t_dataDir	Directory containing the PSF files(results). When specified, this argument will only be used internally and will not alter the currently selected results.The default is the currently selected results directory.

Value Returned

`name`	Name of the simulator.
`nil`	Returns `nil` if could not determine the simulator.

Examples

```
SIMULATOR()
=> "spectre"
SIMULATOR("./simulation/test/hspiceS/schematic")
=> "hspiceS"
```

15

Selection Functions

Selection functions can be very useful if you are responsible for setting up forms for designers. With selection functions, you can make it easier and quicker for a designer to enter schematic data in a type-in field. You give the designer the option of clicking on an item in the schematic window and having the information for that option automatically fill in the field. Typically, selection functions are used as callbacks in code that creates form fields and options.

The following example shows how you might use one of the selection functions as part of some code that creates a *net1* button with an associated type-in field.

```
hiCreateStringField(
    ?name 'net1
    ?prompt "Net Name"
    ?value ""
)
hiCreateButton(
    ?name 'netSelect1
    ?buttonText "Select"
    ?callback "asiSelectNet('net1 ?prompt \"Select net1\")"
)
```

When the designer clicks on the button, the `asiSelectNet` function displays the *"Select net1"* prompt on the schematic window. The designer can click on a net in the schematic window to automatically fill in the type-in field.

Note: Currently there is an *hi* limitation—if the debugger is installed for the session, the prompts are not displayed.

For more information about functions with the *hi* prefix, see the *Cadence User Interface SKILL Reference*.

The next example shows how you might use one of the analysis-specific selection functions as part of some code that creates an *outVsrc* button with an associated type-in field. (This example does not show all the code for adding a noise analysis—it only shows how you might use the `asiSelectAnalysisSource` function.)

```
asiAddAnalysis( tool
    ?name 'noise
    ?prompt "Noise"
        asiCreateAnalysisField(
```

```
            ?name       'outVsrc
            ?prompt     "Output Voltage Source"
        )
        asiCreateAnalysisField(
            ?name       'selectOutVsrc
            ?prompt     "Select"
            ?type       'button
            ?callback       "asiSelectAnalysisSource( 'noise
                                'outVsrc \"Select output voltage
                                source ...\" )"
        )
)
```

When the designer clicks on the *Select* button, the *asiSelectAnalysisSource* function displays the *"Select output voltage source"* prompt on the schematic window. The designer can click on a source in the schematic window to automatically fill in the type-in field.

Note: Typically, the functions in this chapter are for integrators.

asiSelectAnalysisCompParam

```
asiSelectAnalysisCompParam( s_analysisName s_instField s_parField )
     => t | nil
```

Description

Lets the user select a component instance from the schematic to pop up a list box containing the parameters for that instance.

The name of the component automatically fills in the associated form field in the Choosing Analyses form. When the user selects a parameter from the list box, the name of the parameter automatically fills in its associated field.

Arguments

s_analysisName	Name of the analysis.
s_instField	Name of the field to contain the selected component.
s_parField	Name of the field to contain the selected parameter.

Value Returned

t	Returns t if successful.
nil	Returns nil otherwise.

Example

```
asiCreateAnalysisField(
    ?name               'device
    ?prompt             "Component Name"
)
asiCreateAnalysisField(
    ?name               'select
    ?prompt             "Select Component"
    ?type               'button
    ?callback           "asiSelectAnalysisCompParam( 'ac 'device
                            'deviceParam )"
)
asiCreateAnalysisField(
    ?name               'deviceParam
    ?prompt             "Parameter Name"
)
```

Lets the user select a component instance from the schematic in an AC analysis. The name of the component fills in the *device* field, and a list box appears showing all the parameters for that component. Then the user can select a parameter from the list box to fill in the *deviceParam* field.

asiSelectAnalysisInst

```
asiSelectAnalysisInst( s_analysis s_field [t_prompt] )
     => t | nil
```

Description

Lets the user select an instance from the schematic to be used as input for the specified field for the specified analysis. If the user selects any other object, the system ignores the selection and beeps.

Arguments

s_analysis	Analysis type.
s_field	Name of the analysis field.
t_prompt	Prompt string to display in the schematic window. Default Value: "`Select component...`"

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
asiSelectAnalysisInst( 'dc 'component )
```

Lets the user select an instance from the schematic to use as input for the '*component* field of a DC analysis.

asiSelectAnalysisNet

```
asiSelectAnalysisNet( s_analysis s_field [t_prompt] )
     => t | nil
```

Description

Lets the user select a net from the schematic to be used as input for the specified field for the specified analysis. If the user selects any other object, the system ignores the selection and beeps.

Arguments

s_analysis	Analysis type.
s_field	Name of the analysis field.
t_prompt	Prompt string to display in the schematic window. Default Value: "`Select net...`"

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
asiSelectAnalysisNet( 'noise 'outputNode)
```

Lets the user select a net from the schematic to use as input for the '*outputNode* field for a *noise* analysis.

```
asiSelectAnalysisNet( 'xf 'posNode)
```

Lets the user select a net from the schematic to use as input for the '*posNode* field for an *xf* analysis.

asiSelectAnalysisSource

```
asiSelectAnalysisSource( s_analysis s_field [t_prompt])
     => t | nil
```

Description

Lets the user select a source from the schematic to be used as input for the specified field for the specified analysis. If the user selects any other object, the system ignores the selection and beeps.

Note: The property source should be set to `t` in the simInfo of the component to be selected with this function.

Arguments

s_analysis	Analysis type.
s_field	Name of the analysis field.
t_prompt	Prompt string to display in the schematic window. Default Value: "`Select source...`"

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
asiSelectAnalysisSource( 'noise 'outVsrc "Select output voltage source ..." )
```

Lets the user select a source from the schematic to use as input for the '*outVsrc* field for a noise analysis.

```
procedure( almCreateSimInfo_<comp>_<yourSimulator>()
    '( nil
       netlistProcedure    comp
       otherParameters     ( fundname noisefile
                             FNpairs F1 N1 F2 N2 F3 N3 F4 N4 F5 N5
```

```
                          F6 N6 F7 N7 F8 N8 F9 N9 F10 N10 )
      instParameters      ( dc mag phase type xfmag pacmag pacphase val0
                          val1 period delay rise fall width tc1 tc2
                          tnom )
      termOrder           ( PLUS MINUS )
      termMapping         ( nil PLUS \:p MINUS "(FUNCTION minus(root('PLUS')))" )
      propMapping         ( nil dc vdc mag acm phase acp val0 v1 val1 v2
                          period per delay td rise tr fall tf
                          width pw xfmag xfm pacmag pacm
                          pacphase pacp type srcType )
      componentName       vsource
      source              t
   )
)
```

asiSelectInst

```
asiSelectInst(
     s_field
     [?promptt_prompt]
     [?formr_form]
     )
     => t | nil
```

Description

Lets the user select an instance from the schematic. If the user selects any other object, the system beeps and ignores the selection.

Arguments

s_field	Name of the field to be filled in with the selected value.
t_prompt	Prompt string to display in the schematic window. Default Value: "`Select instance...`"
r_form	Form that contains the field to be filled in with the selected value. The default is the form returned by `hiGetCurrentForm`.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
asiSelectInst( 'inst )
```

Lets the user select an instance from the schematic to use as input for the *'inst* field of the current form.

asiSelectNet

```
asiSelectNet(
     s_field
     [?promptt_prompt]
     [?formr_form]
     )
     => t | nil
```

Description

Lets the user select a net from the schematic. If the user selects any other object, the system ignores the selection and beeps.

Arguments

s_field	Name of the field to be filled in with the selected value.
t_prompt	Prompt string to display in the schematic window. Default Value: "`Select net...`"
r_form	Form that contains the field to be filled in with the selected value. The default is the form returned by `hiGetCurrentForm`.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
asiSelectNet( 'net1 )
```

Lets the user select a net from the schematic to use as input for the *'net1* field of the current form.

asiSelectSourceInst

```
asiSelectSourceInst(
     s_field
     [?promptt_prompt]
     [?formr_form]
     )
     => t | nil
```

Description

Lets the user select a source instance on the schematic. If the user selects any other object, the system beeps and ignores the selection.

Arguments

s_field	Name of the field to be filled in with the selected value.
t_prompt	Prompt string to display in the schematic window. Default Value: "`Select source...`"
r_form	Form that contains the field to be filled in with the selected value. The default is the form returned by `hiGetCurrentForm`.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
asiSelectSourceInst( 'srcInst ?prompt "Select voltage source or current source" )
```

Lets the user select a source instance from the schematic to use as input for the *'srcInst* field of the current form.

asiSelectTerm

```
asiSelectTerm(
     s_field
     [?promptt_prompt]
     [?formr_form]
     )
     => t | nil
```

Description

Lets the user select an instance terminal from the schematic. If the user selects any other object, the system ignores the selection and beeps.

Arguments

s_field	Name of the field to be filled in with the selected value.
t_prompt	Prompt string to display in the schematic window. Default Value: "`Select instance terminal...`"
r_form	Form that contains the field to be filled in with the selected value. The default is the form returned by `hiGetCurrentForm`.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
asiSelectTerm( 'term )
```

Lets the user select the instance terminal from the schematic to use as input for the *'term* field of the current form.

asiSelectTermNet

```
asiSelectTermNet(
     s_field
     [?promptt_prompt]
     [?formr_form]
     )
     => t | nil
```

Description

Lets the user select either an instance terminal or a net. If the user selects any other object, the system beeps and ignores the selection.

Arguments

s_field	Name of the field to be filled in with the selected value.
t_prompt	Prompt string to display in the schematic window. Default Value: "`Select terminal or net...`"
r_form	Form that contains the field to be filled in with the selected value. The default is the form returned by `hiGetCurrentForm`.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
asiSelectTermNet( 'termNet )
```

Lets the user select a terminal or a net from the schematic to use as input for the *'termNet* field of the current form.

16

Miscellaneous Functions

This chapter contains additional procedures and methods that can help you integrate your simulator or customize the simulation environment.

ahdlUpdateViewInfo

```
ahdlUpdateViewInfo( t_lib [?cell lt_cell [?view lt_view [?tool lt_tool]]] )
```

Description

Updates cells and cellviews created with releases earlier than 4.4.2 so that the cells and cellviews can use cellview-specific parameters and parameter values. During the update, `ahdlUpdateViewInfo`, parses the Verilog-A or SpectreHDL modules that define the specified cellviews, issues any necessary error messages and updates the cellview CDF information.

Arguments

t_lib	The name of the library to be updated.
lt_cell	An optional name or list of names of cells to be updated. If *lt_cell* is omitted, the function updates every `veriloga` and `ahdl` cellview in the library.
lt_view	An optional name or list of names of cellviews to be updated. If *lt_view* is omitted, the function updates every `veriloga` and `ahdl` cellview associated with the specified cell.
lt_tool	An optional keyed argument added in order to copy the `termMapping` from the tool `simInfo`. If not specified, no copy takes place.

Examples

The first example updates all the `veriloga` and `ahdl` cellviews in a library.

```
ahdlUpdateViewInfo("myLibrary")
```

The next example updates three cells in a library.

```
ahdlUpdateViewInfo("myLibrary" ?cell "res" "cmp" "opamp")
```

The last example updates one specified cellview.

```
ahdlUpdateViewInfo("myLibrary" ?cell "res" ?view "veriloga")
```

amseGeneralSetupForm

```
amseGeneralSetupForm(o_session)
     => t | nil
```

Description

Opens the General Setup form in Virtuoso AMS Designer. The General Setup form appears when you choose *Detailed Setup* and then *General Setup* from the *AMS* menu in Hierarchy Editor.

Arguments

o_session	The simulation session object.

Value Returned

`t`	Displays the General Setup form and returns `t`.
`nil`	Returns `nil` otherwise.

Example

```
amseGeneralSetupForm( session )
```

amseQuickSetupForm

```
amseQuickSetupForm(o_session)
     => t | nil
```

Description

Opens the Quick Setup form in Virtuoso AMS Designer. The Quick Setup form appears when you choose *Quick Setup* from the *AMS* menu in Hierarchy Editor.

Arguments

o_session	The simulation session object.

Value Returned

`t`	Displays the Quick Setup form and returns `t`.
`nil`	Returns `nil` otherwise.

Example

```
amseQuickSetupForm( session )
```

artEnableAnnotationBalloon

```
artEnableAnnotationBalloon(g_value [x_startPoint [x_endPoint]])
     => t | nil
```

Description

Enables or disables the display of parametric sweep results annotated on the schematic. When enabled, the first six result points are displayed in a pop-up window that appears when you hover the mouse pointer over an instance on the schematic. Use `x_firstPoint` and `x_lastPoint` to specify a range of result points to be displayed in the pop-up window.

Arguments

g_value	Enables or disables the pop-up window that displays parametric sweep results annotated on the schematic. Valid Values: `t` enables the pop-up window, `nil` disables the pop-up window. Default Value: `nil`
x_firstPoint	First result point to be displayed in the pop-up window.
x_lastPoint	Last result point to be displayed in the pop-up window. Ensure that the number of result points between *x_firstPoint* and *x_lastPoint* does not exceed six result points because the pop-up window will display only six result points.

Value Returned

`t`	Displays the pop-up window and returns `t`.
`nil`	Returns `nil` otherwise.

Example

```
artEnableAnnotationBalloon(t)
```

Enables the display of parametric sweep results in a pop-up window that appears when you hover the mouse pointer over an instance on the schematic.

```
artEnableAnnotationBalloon(t 15)
```

Displays the 15th result point in a pop-up window, when you hover the mouse pointer over an instance on the schematic.

```
artEnableAnnotationBalloon(t 10 15)
```

Displays the 10th to the 15th result points in a pop-up window, when you hover the mouse pointer over an instance on the schematic.

```
artEnableAnnotationBalloon(nil)
```

Disables the display of parametric sweep results in a pop-up window.

artGenerateHierSymbolCDF

```
artGenerateHierSymbolCDF(d_cellView [g_overwrite] ) => t
```

Description

Creates the cell CDF for the specified cellView, in the same way as happens when you create a symbol from a schematic in Composer. The cellView will be examined for any use of `pPar()` in expressions, and the corresponding parameters will be added to the CDF if they are not already present. Whilst both schematic and symbol cellViews may be passed to this function, it is usually best to pass a schematic cellView, in order to get the `pPar()`s used into the CDF.

Arguments

d_cellView	Database object for the schematic or symbol.
g_overwrite	If the cell CDF already exists, a popup will normally be displayed to ask if you want to overwrite the CDF. If the overwrite argument is specified as non-nil, then the popup is suppressed, and the CDF will be overwritten.

Value Returned

`t`	If it was successful. An error will occur if there were any problems.

Example

```
artGenerateHierSymbolCDF(geGetEditCellView())
```

Recreates the cell CDF for the current cellView.

artGetCdfTargetCV

```
artGetCdfTargetCV() => dbobject | nil
```

Description

This procedure returns the cell view dbobject that is the target of the update instance form or the create instance form. The function requires that either the arm property (`armGet 'cdfInfo 'currentCellView`) is set to a cellView, or that the variable *`cdfgForm`* is bound to a form with a property cellViewId attached (`getq cdfgForm cellViewId`). If neither of these is bound, `nil` is returned.

Argument

None

Value Returned

`dbobject`	Database object.

Examples

```
artGetCdfTargetCV() => db:28458028
artGetCdfTargetCV() => nil
```

artGetCellViewDesignVarList

```
artGetCellViewDesignVarList(d_cellViewId) => l_nameValuePairs
```

Description

Returns the list of design variable name value pairs associated with the top level cellView.

Argument

d_cellViewId	ID of a top level cellview.

Value Returned

l_nameValuePairs	List of name value pairs.

Example

```
artGetCellViewDesignVarList(asiGetTopCellView(asiGetCurrentSession())) =>
(("Rp" "5")
          ("Q" "50")
          ("w" "2*3.14159*1.96e9")
          ("Co" "33p")
          ("Lb" "2.2n")
         )
```

artCurrentInstSimName

```
artCurrentInstSimName( )
     => t_extName
```

Description

This function provides the external instance name for the current instance being formatted in the netlister. The function should only be used for socket netlisters or custom netlist procedures for socket. The routine does not take into consideration case mapping registered for the tool or the prefix for the instance. It does take care of the netlisting mode (FNL, HNL or IHNL).

Argument

None

Value Returned

`t_extName`	Returns the external name for current instance.

Example

```
artCurrentInstSimName()
```

artListToWaveform

```
artListToWaveform( l_xyPairs )
     => o_waveform
```

Description

This function takes a list of X,Y points and translates it into a waveform. Each of the X,Y points is also in a list format. If Y points are represented in a list format, they are treated as complex numbers while creating the waveform.

Argument

l_xyPairs	X, Y points in list format.

Value Returned

o_waveform	Waveform.

Example

```
artListToWaveform( '(( 1 2 ) ( 3 4 ) ( 5 6 )) ) => o_waveform
```

This waveform contains {1,3,5} as X and {2,4,6} as Y points.

artBlankString

```
artBlankString(g_value) => t | nil
```

Description

Returns `true` if the given object is equal to `nil`. If the given object is a string, checks if the string is empty or has blank space characters only and returns `true`. If the object is not `nil` and if the string has non-space characters, it returns `nil`.

Argument

g_value	Data object.

Value Returned

`t`	If the given object is `nil` and also when the object is an empty string or has blank space characters only.
`nil`	Otherwise.

Examples

```
artBlankString("   ")   => t
artBlankString("   h ")  => nil
artBlankString(nil)   => t
```

artMakeString

```
artMakeString( g_anyArg )
     => t_argAsString
```

Description

Converts data of the given data type to a string. The valid data types for the *g_anyArg* argument include symbol, integer, float, and string. The floating point numbers are converted into strings by using the '%.16g' format specification to produce the most precise output.

Argument

g_anyArg	Argument of any data type.

Value Returned

t_argAsString	Returns the string representation of the input argument.
nil	Returns nil otherwise.

Examples

- With symbol data:

```
artMakeString( '\1\2\3 ) => "123"
```

- With integer data:

```
artMakeString( 2 ) => "2"
```

- With float/real data:

```
artMakeString( 2.999 ) => "2.999"
artMakeString( 1e-9 ) => "1e-9"
```

- With list data:

```
artMakeString( '(1 a 2.0 ) ) => "(1 a 2.0 )"
```

- With CDBA inst data:

```
inst = car( geGetSelSet() )
artMakeString( inst ) => "db:123456"
```

- With other data types:

```
artMakeString( nil ) => "nil"
artMakeString( t ) => "t"
```

artMakeStringPrec15

```
artMakeString( g_anyArg )
     => t_argAsString
```

Description

Converts data of the given data type to a string. The valid data types for the *g_anyArg* argument include symbol, integer, float, and string. It similar to the function `artMakeString`, except that it uses the '%.15g' format specification to convert the floating point numbers into strings.

Argument

g_anyArg	Argument of any data type.

Value Returned

t_argAsString	Returns the string representation of the input argument.
`nil`	Returns `nil` otherwise.

Examples

```
artMakeStringPrec15( 2.0 ) => "2.0"
artMakeStringPrec15( 10p ) => "1e-11"
```

asiAddDesignVarList

```
asiAddDesignVarList( o_session l_designVarList )
     => l_newDesignVarList
```

Description

Adds a list of variables to the existing session design variable list.

Arguments

o_session	Simulation session object.
l_designVarList	List of design variables to add to the session design variable list.

Value Returned

l_newDesignVarList	Returns the new list of design variables.

Example

```
asiAddDesignVarList( session '(("rbias" "1k") ("ccap" "1n") ("area" "16u")))
```

Creates a list of name value pairs to add to the list of design variables.

asiAddVerilogArgs

```
asiAddVerilogArgs( o_session t_verilogArg ... )
     => t_verilogArg
```

Description

For a mixed-signal simulation, this method lets you change the list of arguments that are sent to the Verilog-XL simulator with the *-slave* option.

These arguments are the options from the Verilog Options form. Typically, you use this function to add the *+vmxcconfig.vmx* option.

Note for Integrators: This function is defined as a method. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_session	Simulation session object.
t_verilogArg	Arguments (or options) to send to the Verilog-XL simulator with the *-slave* option.

Value Returned

t_verilogArg	Returns the arguments (or options) that are sent to Verilog-XL with the *-slave* option.

Example for Integrators

```
defmethod( asiAddVerilogArgs ( ( session XYZ_session ) verilogArg )
    sprintf( verilogArg "%s +vmxcconfig.vmx" verilogArg )
)
```

Adds *+vmxcconfig.vmx* to the list of Verilog-XL arguments for the XYZ simulator class.

asiLoadState

```
asiLoadState( o_session [?name t_name] [?option t_option] [?stateDir t_stateDir]
     [?lib t_lib] [?cell t_cell] [?simulator t_simulator] )
     => t | nil
```

Description

Loads a saved state into the current simulation environment directly from the CIW without displaying the *Loading State* form.

Arguments

o_session	Simulation session object.
t_name	Name of the state.
t_option	Option to specify whether the state is to be loaded from `'dir` or `'cellview`.
t_stateDir	Directory in which the state is saved. The complete location of the saved state would be *stateDir/lib/cell/simulator*. Specify *stateDir* only when *option* is set to `'dir`.
t_lib	Library name.
t_cell	Cell name.
t_simulator	Name of the simulator. Specify this when *option* is set to `'dir`.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

Examples

```
asiLoadState( session1 ?name "MyState" ?option 'dir ?stateDir "~/.artist_state" )
```

Loads `MyState` directly from the CIW given the directory (`~/.artist_state`) in which the state is saved and the session (`session1`) into which the state is to be loaded.

```
asiLoadState( session2 ?name "spectre_state1" ?option 'cellview ?lib "myLib" ?cell
"myCell" )
```

Loads `spectre_state1` directly from the CIW given the library/cell path (`myLib/myCell`) and the session (`session2`) into which the state is to be loaded.

asiSaveState

```
asiSaveState( o_session [?name t_name] [?option t_option] [?stateDir t_stateDir]
     [?lib t_lib] [?cell t_cell] [?simulator t_simulator]
     [?description t_description] )
     => t | nil
```

Description

Saves the current state of the current simulation environment directly from the CIW without displaying the *Saving State* form.

Arguments

o_session	Simulation session object.
t_name	Name of the state.
t_option	Option to specify whether the state is to be saved in `'dir` or `'cellview`.
t_stateDir	Directory in which the state is to be saved. The complete location of the saved state would be *stateDir/lib/cell/simulator*. Specify *stateDir* only when *option* is set to `'dir`.
t_lib	Library name.
t_cell	Cell name.
t_simulator	Name of the simulator. Specify this when *option* is set to `'dir`.
t_description	Description to be saved with the state.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

Examples

```
asiSaveState( session1 ?name "NewState" ?option 'dir ?stateDir "~/.artist_state"
?description "saved on Jan 1, 2006" )
```

Saves the state `NewState` directly from the CIW given the directory (`~/.artist_state`) and session (`session1`) in which the state is to be saved and the description (`saved on Jan 1, 2006`) to be saved with the state.

asiCheck

```
asiCheck( o_analysis | o_anaOption | o_envOption | o_simOption | o_keepOption
     r_form )
     => t | nil
```

Description

Called by the environment to check values in the analysis and option fields.

Note for Integrators: This procedure is defined as a method for the Analog Class. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_analysis	Specifies the analysis object.
o_anaOption	Specifies the analysis option object.
o_envOption	Specifies the environment option object.
o_simOption	Specifies the simulator option object.
o_keepOption	Specifies the keep option object.
r_form	Form containing the analysis or option fields. This argument is passed into the `asiCheck` method by the environment.

Value Returned

`t`	Returns `t` if the field values are in the correct range.
`nil`	Returns `nil` otherwise.

Example for Integrators

Checks the values of the form fields in an ac analysis.

```
defmethod( asiCheck ( ( ana analog_ac_analysis ) form )

    asiCheckBlankNumericGreater( ana form 'from 0 )
    asiCheckBlankNumericGreater( ana form 'to 0 )

    if( equal( asiGetAnalysisFormFieldVal( form 'ac 'incrType )
```

```
        "Linear" ) then asiCheckBlankNumericGreater( ana form
        'lin 0 )
    else
        asiCheckBlankNumericGreater( ana form 'log 0 )
    )

    asiCheckMultipleGreater( ana form 'to 'from )

)
```

Checks whether the ABSTOL simulator option for Cadence SPICE has a value greater than zero. The Cadence SPICE simulator options class must be defined before the `asiCheck` function is used.

```
defclass( cdsSpice_simOption ( analog_simOption ) ( ) )

defmethod( asiCheck ( ( option cdsSpice_simOption ) form )
    asiCheckBlankNumericGreater( option form 'ABSTOL 0 )
)
```

Note: If you want to write methods for analyses, analysis options, environment options, simulator options, or keep options, you must define the appropriate class in your `classes.il` file.

SKILL Functions to Check Field Values

Listed below are SKILL functions that you can use in code that check field values. For each of these functions, *`o_obj`* can be any one of the following objects:

- *`o_analysis`*
- *`o_anaOption`*
- *`o_envOption`*
- *`o_simOption`*
- *`o_keepOption`*

asiCheckDesignVariable

```
asiCheckDesignVariable( o_obj r_form s_fieldName )
     => t | nil
```

Description

Verifies that the *s_fieldName* entry is a valid design variable.

asiCheckExpression

```
asiCheckExpression( o_obj r_form s_fieldName [ ?subAnaMsg t_subAnaMsg ] )
     => t | nil
```

Description

Verifies that the *s_fieldName* entry is a valid expression.

asiCheckExpressionGreater

```
asiCheckExpressionGreater( o_obj r_form s_fieldName g_value
     [ ?subAnaMsg t_subAnaMsg ] )
     => t | nil
```

Description

Verifies that the *s_fieldName* entry is an expression that evaluates to a value greater than *g_value*.

asiCheckBlankNumeric

```
asiCheckBlankNumeric( o_obj r_form s_fieldName [ ?subAnaMsg t_subAnaMsg ] )
     => t | nil
```

Description

Verifies that the *s_fieldName* entry is a numeric value.

asiCheckBlankNumericGreater

```
asiCheckBlankNumericGreater( o_obj r_form s_fieldName g_value
     [ ?subAnaMsg t_subAnaMsg ] )
     => t | nil
```

Description

Verifies that the *s_fieldName* entry is a numeric value greater than *g_value*.

asiCheckBlankNumericNequal

```
asiCheckBlankNumericNequal( o_obj r_form s_fieldName g_value
     [ ?subAnaMsg t_subAnaMsg ] )
     => t | nil
```

Description

Verifies that the *s_fieldName* entry is a numeric value not equal to *g_value*.

asiCheckBlankNetExists

```
asiCheckBlankNetExists( o_obj r_form s_fieldName [ ?subAnaMsg t_subAnaMsg ] )
     => t | nil
```

Description

Verifies that the *s_fieldName* entry is a valid net name.

asiCheckBlankInstExists

```
asiCheckBlankInstExists( o_obj r_form s_fieldName [ ?subAnaMsg t_subAnaMsg ])
     => t | nil
```

Description

Verifies that the *s_fieldName* entry is a valid instance name.

asiCheckMultipleGreater

```
asiCheckMultipleGreater( o_obj r_form s_largerField s_smallerField )
     => t | nil
```

Description

Verifies that the value of the *s_largerField* entry is greater than that of the *s_smallerField* entry.

Note: These functions are expected to change in later versions of the Virtuoso Analog Design Environment software.

asiCheckSimulationSuccess

```
asiCheckSimulationSuccess( o_session )
     => t | nil
```

Description

Determines if a simulation was successful.

Note for Integrators: This function is defined as a method for the Analog Class. You can overload this method for your simulator class, as shown in the example for integrators.

Argument

o_session	Simulation session object.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
success = asiCheckSimulationSuccess( session )
```

Sets `success` to `t` if the simulation is successful. Sets `success` to `nil` otherwise.

Example for Integrators

```
defmethod( asiCheckSimulationSuccess (( session
        <yourSimulator>_session))
    let( ( success )
    ;; This checks for the existence of the PSF file.
    success = callNextMethod()
    <insert code to perform simulator-specific checking>
    success
    )
)
```

asiCreateLogFileVerilog

```
asiCreateLogFileVerilog( o_session l_entryList )
```

Description

Creates the file logFileVerilog in the .../psf directory to indicate the analysis result of Verilog-XL in a mixed-signal transient run.

Arguments

o_session	Virtuoso Analog Design Environment session.
l_entry_list	Must be in the form.
nil	If there is no transient analysis.
'(("<transient-analysis-instance-name>" "<transient-analysis-type-name>"))	If there is a transient analysis and digital results are available.

Value Returned

None

Examples

For cdsSpiceVerilog:

```
asiCreateLogFileVerilog( o_session '(("timeSweep" "tran")) )
```

For spectreVerilog:

```
asiCreateLogFileVerilog( o_session '(("tran-tran" "tran")) )
```

For spectreSVerilog:

```
asiCreateLogFileVerilog( o_session '(("timeSweep-tran" "tran")) )
```

When transient analysis is not performed or when no digital results are saved for Verilog:

```
asiCreateLogFileVerilog( o_session nil)
```

asiDcStore

```
asiDcStore( o_session t_fileName )
     => t | nil
```

Description

Copies the DC node voltages in *processId*.dc to the name you pass in *fileName*. This function assumes that your simulator writes the DC node voltages to netlistDir/raw/*processId*.dc.

Note for Integrators: This function is defined as a method for the Analog Class. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_session	Specifies the session object.
t_fileName	Name of the file in which to store the DC node voltages.

Value Returned

t	Returns t if the DC voltages are stored in *fileName*.
nil	Returns nil otherwise.

Example for Integrators

```
defmethod( asiDcStore ( ( session XYZ_session ) fileName )
    let( ( netlistDir processId str )
        netlistDir = asiGetNetlistDir(session)
        processId = asiGetSimProcessId(session)
        str = strcat( "/bin/cp " netlistDir "/raw/" processId ".dc "
            fileName )
        system( str )
        t
    )
)
```

asiGetCurrentSession

```
asiGetCurrentSession( )
     => o_session | nil
```

Description

Returns the session object for the current session.

Note: Use this procedure only from within menu callbacks to get the current session.

Argument

None

Value Returned

o_session	Returns the session object for the current session.
nil	Returns nil if there is no current session defined.

asiGetDesignVarList

```
asiGetDesignVarList( o_session )
     => l_designVarList | nil
```

Description

Gets the list of design variables for the design associated with the session you specify.

Argument

o_session	Simulation session object.

Value Returned

l_designVarList	Returns the list of design variables.
nil	Returns nil if there are no design variables associated with the specified session.

Example

```
designVarList = asiGetDesignVarList( session )
```

Returns the list of design variables in designVarList.

asiGetFormFieldChoices

```
asiGetFormFieldChoices( r_form s_fieldName )
     => l_choices | nil
```

Description

Gets the list of choices for a form field that is set up as a list box.

Arguments

r_form	Form containing the field. The form can be one of the following: Environment options Simulator options Keep options Analysis options
s_fieldName	Name of a field of the type `listBox`.

Value Returned

l_choices	Returns the list of choices for the field.
`nil`	Returns `nil` if the field cannot be accessed.

Related Function

To get the list of choices for a field on the Choosing Analyses form, see the asiGetAnalysisFormFieldChoices function on page 9-356.

asiGetFormFieldVal

```
asiGetFormFieldVal( r_form s_fieldName )
     => g_value | nil
```

Description

Gets the value of a field on a form.

Arguments

r_form	Form containing the field. The form can be one of the following: Environment options Simulator options Keep options Analysis options
s_fieldName	Name of the field.

Value Returned

g_value	Returns the value of the field.
nil	Returns nil if the field cannot be accessed.

Example

```
asiGetFormFieldVal( form 'RELTOL )
```

Gets the value of the RELTOL field on the Simulator Options form.

Related Function

To get the value of a field on the Choosing Analyses form, see the asiGetAnalysisFormFieldVal function on page 9-357.

asiGetKeepList

```
asiGetKeepList( o_session )
     => l_keepList | nil
```

Description

Gets a list of signals and currents that are saved during simulation.

Argument

o_session	Simulation session object.

Value Returned

l_keepList	Returns the list of signals and currents to be saved during simulation.
nil	Returns nil if no signals were specified to be kept during simulation.

Example

```
keeplist = asiGetKeepList( session )
```

Returns the list of signals and currents for a given session in keeplist.

asiGetLogFileList

```
asiGetLogFileList( o_session )
     => l_logFiles | nil
```

Description

Returns a list of the names of the log files.

Note for Integrators: This procedure is defined as a method for the Analog Class. You can overload this method to return your list of log files, as shown in the example for integrators.

Argument

o_session	Session object.

Value Returned

l_logFiles	Returns the list of log files. For analog-only simulation, the following list is returned:
	`("logFile")`
	For mixed-signal simulation, the following list is returned:
	`( "logFile" "logFileVerilog" )`

`nil`	Returns `nil` if there is an error.

Example

```
asiGetLogFileList( asiGetCurrentSession())
=> ("logFile")
```

Returns the list of log files for the session.

Example for Integrators

```
defmethod( asiGetLogFileList ( ( _session <yourSimulator>_session ) )
    '( "<yourSimulator>logFile" )
)
```

Overloads the `asiGetLogFileList` function to return the log file for your simulator.

asiGetMarchList

```
asiGetMarchList( o_session )
     => l_marchList | nil
```

Description

Returns a list of signals that are to be marched during simulation.

Argument

o_session	Simulation session object.

Value Returned

l_marchList	Returns the list of signals to march during simulation.
nil	Returns nil if no signals were specified to be marched.

Example

```
marchList = asiGetMarchList( session )
```

Returns the list of march signals for the session in marchList.

asiGetNetlistDir

```
asiGetNetlistDir( o_session )
     => t_netlistDir
```

Description

Returns the netlist directory. If the directory does not exist, this function creates it.

Argument

o_session	Simulation session object.

Value Returned

t_netlistDir	Returns the path name for the netlist directory.

Example

```
netlistDir=asiGetNetlistDir(session)
```

Returns the netlist directory.

asiGetOutputList

```
asiGetOutputList( o_session )
     => o_outputsList | nil
```

Description

Returns a list of structures. Each structure defines a output to be saved and/or plotted after simulation.

Argument

o_session	Specifies an analysis object.

Value Returned

o_outputsList	Returns a list of structures. Each structure defines one output.
nil	Returns `nil` if there was an error.

Example

After opening an ADE session and setting up outputs, in the CIW:

```
s=asiGetCurrentSession()
=>stdobj@0x1d14060
outs=asiGetOutputList(s)
=>(sevOutputStruct@0x15e71b8 sevOutputStruct@0x15e7208 sevOutputStruct@0x153ef48)
outs~>signal
=>("/out" "/net15" nil)
outs~>expression
=>(nil nil ymax(VT("/net15")))
```

asiGetPlotList

```
asiGetPlotList( o_session )
     => l_plotList | nil
```

Description

Gets the list of signals that can be plotted for the current simulation session.

Argument

o_session	Simulation session object.

Value Returned

l_plotList	Returns the list of the items that can be plotted.
nil	Returns nil if no signals were specified to be plotted.

Example

```
plotList = asiGetPlotList( session )
```

Returns the list of items that can be plotted for the given session in plotList.

asiGetPsfDir

```
asiGetPsfDir( o_session )
     => t_dirName | nil
```

Description

Returns the name of the PSF directory.

Argument

o_session	Session object.

Value Returned

t_dirName	Returns the name of the PSF directory.
nil	Returns nil if there is an error.

Example

```
psfDir = asiGetPsfDir( session )
```

Returns the PSF directory.

asiGetSession

```
asiGetSession(
     { x_id | w_window | r_form | s_name | o_analysis }
     )
     => o_session | nil
```

Description

Returns the session object given one of five possible identifiers.

Arguments

x_id	Integer identifier for the session.
w_window	Encapsulation window for the session.
r_form	Form for the session, which must be one of the following: Analyses Environment options Simulator options Keep options Analysis options
s_name	Symbolic name for the session, for example, *spectre0*.
o_analysis	Analysis object.

Value Returned

o_session	Returns the session object if successful.
nil	Returns nil otherwise.

Example

```
session = asiGetSession( 'spectre0 )
```

Returns the session object for the *spectre0* session.

asiGetSimName

```
asiGetSimName( { o_tool | o_session } )
     => t_simulatorName
```

Description

Gets the name of the simulator for a tool or session object.

Arguments

o_tool	Simulation tool object.
o_session	Simulation session object.

Value Returned

t_simulatorName	Returns the name of the simulator.

Example

```
simulatorName = asiGetSimName( session )
```

Returns the simulator name for the given session in `simulatorName`.

asiGetTool

```
asiGetTool( { ts_toolName | o_session } )
     => o_tool
```

Description

Returns the tool object associated with the specified tool name or session. If the tool object is not found, an attempt is made to create and initialize the tool.

Arguments

ts_toolName	Name of the tool.
o_session	Session object.

Value Returned

o_tool	Returns the tool object.

Example

```
tool = asiGetTool( 'XYZ )
```

Returns the tool object for the XYZ simulator.

asiGetTopCellView

```
asiGetTopCellView( o_session )
     => d_cellView | nil
```

Description

Returns the top-level cellview associated with the session.

Arguments

o_session	Session object.

Value Returned

d_cellView	Returns the top-level cellview.
nil	Returns nil if there is an error.

Example

```
cv = asiGetTopCellView( session )
lib = cv~>libName
cell = cv~>cellName
view = cv~>viewName
```

Returns the library, cell, and view names.

asiSendSim

```
asiSendSim( o_session t_command t_callback g_param g_saveOutput )
     => t
```

Description

Sends a command to Cadence SPICE to forward to the target simulator.

Arguments

o_session	Session object.
t_command	Command (string) to send to Cadence SPICE.
t_callback	Routine to call when this command terminates. Frequently, this argument is set to nil. Callback parameter list: (*l_list g_param*) *l_list* is a list of strings (one string per line) that are generated by Cadence SPICE as output from *t_command*.

Note: *l_list* is generated only if *g_saveOutput* is non-nil.

g_param	Parameter for the callback routine. Frequently, this argument is set to nil.
g_saveOutput	Boolean flag that specifies whether or not the output from the command you sent to Cadence SPICE is saved in *l_list*. If you specify this argument as non-nil, the output is saved in *l_list*; otherwise, the output is not saved.

Value Returned

t	Returns t when the command is sent to Cadence SPICE.

Example

```
asiSendSim( session "restore off" nil nil nil )
```

Sends the "restore off" command to Cadence SPICE to be forwarded to the target simulator.

asiSetDesignVarList

```
asiSetDesignVarList( o_session l_designVarList )
     => l_designVarList
```

Description

Sets the design variable list for a session.

This function is used to set the design variable list (a list of (*varName*, *varVal*) pairs) for a session, stripping of any invalid *varName* (the one that does not match the regular expression `^[a-zA-Z_][a-zA-Z_0-9]*$`).

Arguments

o_session	Simulator session object.
l_designVarList	List of design variables you want to use. The list should be comprised of one or more 2 element sub-lists of the following format: `("variableName" "value")` If the list is `'nil'`, all design variables will be removed from the current session.

Value Returned

l_designVarList	Returns the list of design variables for the session.

Example

```
varList = '(("var1" "100") ("var2" "ABC"))
asiSetDesignVarList(asiGetCurrentSession() varList)
=> (("var2" "ABC") ("var1" "100"))
```

Here, `var1` & `var2` are valid variable names starting with a-z or A-Z characters.

If you want add more design variables you can use `asiAddDesignVarList`:

```
    varList = '(("1var1" "100") ("var3" "ABC"))
    asiAddDesignVarList(asiGetCurrentSession() varList)
=>*Error* asiSetDesignVarList: bad name "1var1", not added
=> (("var3" "ABC") ("var2" "ABC") ("var1" "100"))
```

Here `var3` is a valid variable name and therefore gets added to the design variable list, but `1var1` is discarded as it is not a valid variable name (not starting with a-z or A-Z character).

asiSetFormFieldChoices

```
asiSetFormFieldChoices( r_form s_fieldName l_choices )
     => l_choices | nil
```

Description

Sets the list of choices to appear in the list box for the specified form field.

Arguments

r_form	Form containing the field. The form can be one of the following: Environment options Simulator options Keep options Analysis options
s_fieldName	Name of a field of the type `listBox`.
l_choices	List of choices for the field.

Value Returned

l_choices	Returns the new list of choices for the field.
`nil`	Returns `nil` if the field cannot be accessed.

Related Function

To set the list of choices for a field on the Choosing Analyses form, see theasiSetAnalysisFormFieldChoices function on page 9-372.

asiSetFormFieldVal

```
asiSetFormFieldVal( r_form s_fieldName g_value )
     => g_value | nil
```

Description

Sets the value of a field on a form.

Arguments

r_form	Form containing the field. The form can be one of the following: Environment options Simulator options Keep options Analysis options
s_fieldName	Name of the field.
g_value	Value for the field.

Value Returned

g_value	Returns the new value for the field.
`nil`	Returns `nil` if the field cannot be accessed.

Example

```
asiSetFormFieldVal( form 'RELTOL 1e-2 )
```

Sets the RELTOL field on the Simulator Options form to 1e-2.

Related Function

To set the value of a field on the Choosing Analyses form, see the asiSetAnalysisFieldVal function on page 9-371.

asiSetKeepList

```
asiSetKeepList( o_session l_KeepList )
     => l_KeepList
```

Description

Sets the list of the specific signals and currents to save during simulation.

Note: If you want to save *all* the outputs of a particular type, see Chapter 12, "Keep Option Functions."

Arguments

o_session	Simulation session object.

l_KeepList	This is a list of selected signals (nets) or terminals comprising of the following information:
	`window_id`
	`libName`
	`CellName`
	`ViewName`

Value Returned

l_KeepList	Returns the list of signals and currents to be saved during simulation.

Example

```
asiSetKeepList( session myKeepList )
```

Sets `myKeepList` as the list of signals to be saved for the given session.

asiSetMarchList

```
asiSetMarchList( o_session l_MarchList )
     => l_MarchList
```

Description

Sets the list of signals to march during simulation.

Arguments

o_session	Simulation session object.

l_MarchList	List of signals to march during simulation.
	This is a list of selected signals (nets) or terminals comprising of the following information:
	`window_id`
	`libName`
	`CellName`
	`ViewName`

Value Returned

l_MarchList	Returns the list of signals to march during simulation.

Example

```
asiSetMarchList( session myMarchList )
```

Sets `myMarchList` as the list of signals to march for the given session.

asiSetPlotList

```
asiSetPlotList( o_session l_PlotList )
     => l_PlotList
```

Description

Sets the plot list for the current simulation session.

Arguments

o_session	Simulation session object.

l_PlotList	List of items to plot in this simulation session.
	This is a list of selected signals (nets) or terminals comprising of the following information:
	`window_id`
	`libName`
	`CellName`
	`ViewName`

Value Returned

l_PlotList	Returns the list of items that can be plotted.

Example

```
asiSetPlotList( session myPlotList )
```

Sets the plot list for the given session.

asiSetSyncFlag

```
asiSetSyncFlag( g_flag ) => g_flag
```

Description

This function makes the simulation run in a blocking mode. This implies that control does not come back till the simulation is over. This functionality is needed in the replay files in order to wait for a simulation to finish before proceeding. Call this function (`asiSetSyncFlag(t)`) just before launching a simulation. This function should only be used for testing capabilities. OCEAN should be used for scripting simulations.

Arguments

g_flag (`t/nil`)	To synchronise the simulation.

Values Returned

`t`	When the function sets the value as `true`.
`nil`	Otherwise.

asiTransientStore

```
asiTransientStore( o_session t_fileName )=> t | nil
```

Description

Copies the final transient operating points in *processId*.tr to the name you pass in fileName. This function assumes that your simulator writes the final transient operating points to netlistDir/raw/*processId*.tr.

Note for Integrators: This function is defined as a method for the Analog Class. You can overload this method for your simulator class, as shown in the example for integrators.

Arguments

o_session	Specifies the session object.
t_fileName	Name of the file in which to store the transient node voltages.

Value Returned

t	Returns t if the transient node voltages are stored in fileName.
nil	Returns nil otherwise.

Example for Integrators

```
defmethod( asiTransientStore (( session <yourSimulator>_session ))
    <insert any code you need>
)
```

Note: See the asiDcStore function for similar example that is more complete.

asiMapNetName

```
asiMapNetName( t_dataDir l_specifier )
     => l_specifier
```

Description

Maps the hierarchical schematic net name to the name in the netlist.

Arguments

t_dataDir	Specifies the data directory.
l_specifier	Lists the name of the net to map.

Value Returned

l_specifier	Returns the list containing the mapped name.

Note: The list *l_specifier* passed as the argument is also destructively modified to contain the mapped name.

asiMapTerminalName

```
asiMapTerminalName( t_dataDir l_specifier )
     => l_specifier
```

Description

Maps the hierarchical schematic terminal name to the name in the netlist.

Arguments

t_dataDir	Specifies the data directory.
l_specifier	Lists the name of the terminal to map.

Value Returned

l_specifier	Returns the list containing the mapped name.

Note: The list *l_specifier* passed as the argument is also destructively modified to contain the mapped name.

asiMapInstanceName

```
asiMapInstanceName( t_dataDir l_specifier )
     => l_specifier
```

Description

Maps the hierarchical schematic instance name to the name in the netlist.

Arguments

t_dataDir	Specifies the data directory.
l_specifier	Lists the name of the instance to map.

Value Returned

l_specifier	Returns the list containing the mapped name.

Note: The list *l_specifier* passed as the argument is also destructively modified to contain the mapped name.

asiRegCallBackOnSimComp

```
asiRegCallBackOnSimComp( o_session t_callback )
     => t | nil
```

Description

Registers the specified user-defined callback function to run after completion of a simulation. The callback function should accept two arguments, a session object and the simulation status.

Note: In case of a simulation with multiple points, the callback is executed multiple times.

The registered callback function is executed only for local simulations.

Arguments

o_session	Simulation session object.
t_callback	Name of the callback function to be registered.

Value Returned

t	Returns t when the callback function is registered successfully.
nil	Otherwise, returns nil.

Example

```
asiRegCallBackOnSimComp(asiGetCurrentSession 'myfunc)
```

In this example, the myfunc function is registered to run after completion of a simulation. You can define myfunc as:

```
(defmethod myfunc (session status)
    ....
)
```

asiUnRegCallBackOnSimComp

```
asiUnRegCallBackOnSimComp( t_callback )
     => t | nil
```

Description

Deregisters the specified callback function registered to run on completion of the simulation.

Arguments

t_callback	Name of callback function to be deregistered.

Value Returned

`t`	Returns `t` if the callback is deregistered successfully.
`nil`	Otherwise, returns `nil`.

Example

```
asiUnRegCallBackOnSimComp(asiGetCurrentSession 'myfunc)
```

In this example, the `myfunc` function is deregistered.

almDefineParam_accuracyMode

```
almDefineParam_accuracyMode(t_cellName)
     => s_cellParameter | nil
```

Description

This function is used to define the accuracy mode.

Arguments

t_cellName	Name of the cell.

Value Returned

s_cellParameter	The cell parameter.
nil	Returns nil otherwise.

Example

```
almDefineParam_accuracyMode(nport)
```

almDefineParam_additionalParam

```
almDefineParam_additionalParam(t_cellName)
     => s_cellParameter | nil
```

Description

This function is used to enable additional parameters.

Arguments

t_cellName	Name of the cell.

Value Returned

s_cellParameter	The cell parameter.
nil	Returns nil otherwise.

Example

```
almDefineParam_additionalParam(nport)
```

almDefineParam_fq

```
almDefineParam_fq(t_cellName)
     => s_cellParameter | nil
```

Description

This function is used to define a `fq` parameter.

Arguments

t_cellName	Name of the cell.

Value Returned

s_cellParameter	The cell parameter.
`nil`	Returns `nil` otherwise.

Example

```
almDefineParam_fq(indq)
```

almDefineParam_noiseParaLabel

```
almDefineParam_noiseParaLabel(t_cellName)
     => s_cellParameter | nil
```

Description

Noise parameter label.

Arguments

t_cellName	Name of the cell.

Value Returned

s_cellParameter	The cell parameter.
nil	Returns nil otherwise.

Example

```
almDefineParam_noiseParaLabel(nport)
```

almDefineParam_nportFileB

```
almDefineParam_nportFileB(t_cellName)
     => s_cellParameter | nil
```

Description

`nport` file.

Arguments

t_cellName	Name of the cell.

Value Returned

s_cellParameter	The cell parameter.
`nil`	Returns `nil` otherwise.

Example

```
almDefineParam_nportFileB(nport)
```

almDefineParam_otherParaLabel

```
almDefineParam_otherParaLabel(t_cellName)
     => s_cellParameter | nil
```

Description

This function is used to enable other paramters.

Arguments

t_cellName	Name of the cell.

Value Returned

s_cellParameter	The cell parameter.
nil	Returns nil otherwise.

Example

```
almDefineParam_otherParaLabel(nport)
```

almDefineParam_tranAdvanParaLabel

```
almDefineParam_tranAdvanParaLabel(t_cellName)
     => s_cellParameter | nil
```

Description

Advanced `tran` parameter.

Arguments

t_cellName	Name of the cell.

Value Returned

s_cellParameter	The cell parameter.
`nil`	Returns `nil` otherwise.

Example

```
almDefineParam_tranAdvanParaLabel(nport)
```

almDefineParam_tranParaLabel

```
almDefineParam_tranParaLabel(t_cellName)
     => s_cellParameter | nil
```

Description

`tran` parameter label.

Arguments

t_cellName	Name of the cell.

Value Returned

s_cellParameter	The cell parameter.
`nil`	Returns `nil` otherwise.

Example

```
almDefineParam_tranParaLabel(nport)
```

almGetModuleName

```
almGetModuleName(t_lib t_cell [?view t_view] [?tool t_tool])
     => s_moduleName | nil
```

Description

Returns the module name for the arguments specified. The netlist procedure is set with `almSetModuleName`.

Arguments

t_lib	Name of the library which the cell resides.
t_cell	Name of the cell.
t_view	Name of the view.
t_tool	Name of the tool. If the *t_view* argument has been specified, and any view-specific information is available for this view, the *t_tool* argument is ignored.

Value Returned

s_moduleName	The module name, if one is defined.
`nil`	Returns `nil` otherwise.

Example

```
almGetModuleName("analogLib" "vdc" ?tool "spectre")
```

almGetNamePrefix

```
almGetNamePrefix(t_lib t_cell [?view t_view] [?tool t_tool])
     => t_namePrefix | nil
```

Description

Returns the name prefix for the arguments specified. The netlist procedure is set with `almSetNamePrefix`.

Arguments

t_lib	Name of the library which the cell resides.
t_cell	Name of the cell.
t_view	Name of the view.
t_tool	Name of the tool. If the *t_view* argument has been specified, and any view-specific information is available for this view, the *t_tool* argument is ignored.

Value Returned

t_namePrefix	The name prefix, if one is defined.
`nil`	Returns `nil` otherwise.

Example

```
almGetNamePrefix("analogLib" "npn" ?tool "spectreS")
```

almGetParameterList

```
almGetParameterList(t_lib t_cell [?view t_view] [?tool t_tool] [?entry s_entry])
     => l_parameterList | nil
```

Description

Returns the list of parameters for the arguments specified. The parameter list is set with `almSetParameterList`.

Arguments

t_lib	Name of the library which the cell resides.
t_cell	Name of the cell.
t_view	Name of the view.
t_tool	Name of the tool. If the *t_view* argument has been specified, and any view-specific information is available for this view, the *t_tool* argument is ignored.
s_entry	Parameter entry. The default value is `instParameters`.

Value Returned

l_parameterList	The list of parameter names, when defined.
`nil`	Returns `nil` otherwise.

Example

```
almGetParameterList("analogLib" "vdc" ?tool "spectre" ?entry "instParameters")
```

almGetTerminalList

```
almGetTerminalList(t_lib t_cell [?view t_view] [?tool t_tool])
     => l_terminalList | nil
```

Description

Returns the list of terminal names for the arguments specified. The netlist procedure is set with `almSetTerminalList`.

Arguments

t_lib	Name of the library which the cell resides.
t_cell	Name of the cell.
t_view	Name of the view.
t_tool	Name of the tool. If the *t_view* argument has been specified, and any view-specific information is available for this view, the *t_tool* argument is ignored.

Value Returned

l_terminalList	The list of terminal names, when defined.
`nil`	Returns `nil` otherwise.

Example

```
almGetTerminalList("analogLib" "vdc" ?tool "spectre")
```

almGetTerminalMap

```
almGetTerminalMap(t_lib t_cell t_terminal [?view t_view] [?tool t_tool])
     => s_moduleName | nil
```

Description

Returns the simulator name of a terminal for the arguments specified. The netlist procedure is set with `almSetTerminalMap`.

Arguments

t_lib	Name of the library which the cell resides.
t_cell	Name of the cell.
s_terminal	Name of the terminal for which the mapped name is returned.
t_view	Name of the view.
t_tool	Name of the tool. If the *t_view* argument has been specified, and any view-specific information is available for this view, the *t_tool* argument is ignored.

Value Returned

s_map	Mapped name of the terminal, when one is available.
`nil`	Returns `nil` otherwise.

Example

```
almGetTerminalMap("analogLib" "vdc" "PLUS" ?tool "spectre")
```

almSetTerminalMap

almSetTerminalMap(*t_lib t_cell t_name t_map* [?view *t_view]* [?tool *t_tool]*)

Description

Sets the mapped name (simulator name) for a terminal name (schematic name) for the arguments specified. The mapped name is used for results display and it is used by the simulator interface in the simulator input file.

Arguments

t_lib	Name of the library in which the cell resides.
t_cell	Name of the cell.
t_name	Schematic name of the terminal.
t_map	Mapped simulator name of the terminal.
t_view	Name of the view.
t_tool	Name of the tool. This argument is ignored when the *t_view* argument is specified.

Example

```
almSetTerminalMap("analogLib" "res" "PLUS" ":1" ?tool 'spectre)
```

almGetOpPointParamMap

```
almGetOpPointParamMap( t_lib t_cell [?view t_view] [?tool t_tool]) =>
     l_opMap | nil
```

Description

Returns the operating-point parameter map for the arguments specified. The netlist procedure is set with `almSetOpPointParamMap`.

Arguments

t_lib	Name of the library which the cell resides.
t_cell	Name of the cell.
t_view	Name of the view.
t_tool	Name of the tool. If the *t_view* argument has been specified, and any view-specific information is available for this view, the *t_tool* argument is ignored.

Value Returned

l_opMap	Operating-point parameter map, if one is defined.
`nil`	Returns `nil` otherwise.

Example

```
almGetOpPointParamMap( "analogLib" "npn" ?tool "spectre" )
```

almSetOpPointParamMap

almSetOpPointParamMap(*t_lib t_cell l_map* [?view *t_view]* [?tool *t_tool]*)

Description

Sets the operating-point parameter map for the arguments specified. This map is used in results display.

Arguments

t_lib	Name of the library in which the cell resides.
t_cell	Name of the cell.
l_map	Operating-point map.
t_view	Name of the view.
t_tool	Name of the tool. This argument is ignored when the *t_view* argument is specified.

Example

```
almSetOpPointParamMap( "analogLib" "npn" nil ?tool 'spectre )
```

almGetNetlistProcedure

```
almGetNetlistProcedure(t_lib t_cell [?view t_view] [?tool t_tool]) =>
     s_procedure | nil
```

Description

Returns the netlist procedure for the arguments specified. The netlist procedure is used to netlist an instance. The netlist procedure is declared with `almSetNetlistProcedure`.

Arguments

t_lib	Name of the library which the cell resides.
t_cell	Name of the cell.
t_view	Name of the view.
t_tool	Name of the tool. If the *t_view* argument has been specified, and any view-specific information is available for this view, the *t_tool* argument is ignored.

Value Returned

s_procedure	Returns the netlist procedure, if one is defined.
`nil`	Returns `nil` if there is an error.

Example

```
almGetNetlistProcedure("analogLib" "vdc" ?tool "spectre")
```

almGetViewInfoNameList

```
almGetViewInfoNameList(t_lib t_cell)
     => l_view_list | nil
```

Description

Returns the list of view-name strings for which view-specific information is available, and for which the view exists.

Arguments

t_lib	Name of the library in which the cell resides.
t_cell	Name of the cell.

Value Returned

l_view_list	A list of strings which are the names of the views that have view-specific information.
nil	If no views have view-specific information.

Example

```
almGetViewInfoNameList( "rfLib" "balun" )
```

almGetNetlistType

```
almGetNetlistType(t_lib t_cell [?view t_view] )
     => t_netlistType | nil
```

Description

Returns the netlist type for the library, cell and view specified, if view-specific information is available. This procedure issued during netlisting.

Arguments

t_lib	Name of the library in which the cell resides.
t_cell	Name of the cell.
t_view	Name of the view.

Value Returned

t_netlistType	Netlist type for the view or the tool specified, if the view has view-specific information.
nil	Returns nil if no view-specific information is available.

Example

```
almGetNetlistType( "rfLib" "balun" ?view "veriloga")
```

almHasViewInformation

```
almHasViewInformation( t_lib t_cell t_view )
     => t | nil
```

Description

Determines if the view-specific information is available for the library, cell, and view specified.

Arguments

t_lib	Name of the library in which the cell resides.
t_cell	Name of the cell.
t_view	Name of the view.

Value Returned

`t`	Returns `t` when the view-specific information is available for the lib, cell, and view specified.
`nil`	Returns `nil` otherwise.

Example

```
almHasViewInformation("rfLib" "balun"  "veriloga")
```

almSetNamePrefix

```
almSetNamePrefix(t_lib t_cell t_name [?view t_view] [?tool t_tool])
```

Description

Sets the name prefix for the view or the tool specified.

The name prefix is used for the instance name in the netlist. The use of the name prefix in the netlist depends on the simulator interface. If the simulator requires that the instance name of a type of components starts with a certain prefix, then the interface must use this name prefix. If this is not a requirement for a simulator, then the interface should not use the name prefix. For the spectre interface, the name prefix is ignored.

Arguments

t_lib	Name of the library in which the cell resides.
t_cell	Name of the cell.
t_name	Name of the prefix.
t_view	Name of the view.
t_tool	Name of the tool. This argument is ignored when the *t_view* argument is specified.

Example

```
almSetNamePrefix (“analogLib” “npn” “analog” "Q” ?tool "spectreS")
```

almSetModuleName

```
almSetModuleName (t_lib t_cell t_name [?view t_view] [?tool t_tool] )
```

Description

Sets the module name for the view or tool.

The module name, also referred to as component name or model name, is used in the netlist to identify the type of component.

Arguments

t_lib	Name of the library in which the cell resides.
t_cell	Name of the cell.
t_name	Module name.
t_view	Name of the view.
t_tool	Name of the tool. This argument is ignored when the *t_view* argument is specified.

Example

```
almSetModuleName("analogLib" "res" "resistor" ?tool "spectre")
```

almSetNetlistProcedure

```
almSetNetlistProcedure(t_lib t_cell s_procedure [?view t_view] [?tool t_tool] )
```

Description

Sets the netlist procedure for the arguments specified.

The netlist is used to netlist an instance.

Arguments

t_lib	Name of the library in which the cell resides.
t_cell	Name of the cell.
s_procedure	Name of the netlist procedure.
t_view	Name of the view.
t_tool	Name of the tool. This argument is ignored when the *t_view* argument is specified.

Example

```
almSetNetlistProcedure( "analogLib" "vdc" 'spectreSrcPrim ?tool "spectre")
```

almSetParameterList

```
almSetParameterList(t_lib t_cell l_parameter [?view t_view] [?tool t_tool]
     [?entry s_entry])
```

Description

Sets list of parameter names for the arguments specified.

These parameters are printed to the netlist if the user has given these values.

Arguments

t_lib	Name of the library in which the cell resides.
t_cell	Name of the cell.
l_parameter	List of parameter names. Each element must be a symbol.
t_view	Name of the view.
t_tool	Name of the tool. This argument is ignored when the *t_view* argument is specified.
s_entry	Parameter entry. The default value is `instParameters`.

Example

```
almSetParameterList( "analogLib" "npn" '(area m trise region) ?tool "spectre"
?entry "instParameters")
```

almSetTerminalList

```
almSetTerminalList( t_lib t_cell l_termList [?view t_view] [?tool t_tool] )
```

Description

Sets the list of terminal names for the lib arguments specified.

The simulator names of the signals connected to these terminals are printed to the netlist, according to the order of this terminal list.

Arguments

t_lib	Name of the library in which the cell resides.
t_cell	Name of the cell.
l_termList	Terminal list.
t_view	Name of the view.
t_tool	Name of the tool. This argument is ignored when the *t_view* argument is specified.

Example

```
almSetTerminalList( "analogLib" "npn" '(C B E) "vdc" ?tool "spectre")
```

almSetPropMappingList

```
almSetPropMappingList( t_lib t_cell l_parameter [?view t_view] [?tool t_tool]
     [?entry s_entry] )
```

Description

Sets the list of propMapping for the arguments specified.The parameters specified in the *l_parameter* list are mapped and printed to the netlist if the user has specified these values.

Arguments

t_lib	Name of the library in which the cell resides.
t_cell	Name of the cell.
l_parameter	List of parameter names. Each element must be a symbol.
t_view	Name of the view.
t_tool	Name of the tool. This argument is ignored when the *t_view* argument is specified.
s_entry	Parameter entry. The default value is `"propMapping"`.

Example

```
almSetPropMappingList("analogLib" "mind" '(nil coupling K1) ?tool "spectre" ?entry
"propMapping")
```

almGetPropMappingList

```
almGetPropMappingList( t_lib t_cell [?view t_view] [?tool t_tool] [?entry s_entry]
     )=> l_propMappingList | nil
```

Description

Returns the `propMapping` parameter list for the arguments specified. The propMapping list is set with `almSetPropMappingList`.

Arguments

t_lib	Name of the library in which the cell resides.
t_cell	Name of the cell.
t_view	Name of the view.
t_tool	Name of the tool. This argument is ignored when the *t_view* argument is specified.
s_entry	Parameter entry. The default value is `"propMapping"`.

Value Returned

`l_propMappingList`	The list containing propMapping names, when defined.
`nil`	Returns nil otherwise.

Example

```
almGetPropMappingList("analogLib" "vdc"  ?tool "spectre" ?entry "propMapping")
```

almSetOtherParameterList

```
almSetOtherParameterList( t_lib t_cell l_parameter  [?view t_view] [?tool t_tool]
     [?entry s_entry] )
```

Description

Sets list of other parameter names for the arguments specified.

Arguments

t_lib	Name of the library in which the cell resides.
t_cell	Name of the cell.
l_parameter	List of parameter names. Each element must be a symbol
t_view	Name of the view.
t_tool	Name of the tool. This argument is ignored when the *t_view* argument is specified.
s_entry	Parameter entry. The default value is "otherParameters".

Example

```
almSetOtherParameterList("analogLib" "mind" '(ind1 ind2)  ?tool "spectre" ?entry
"otherParameters")
```

almGetOtherParameterList

```
almGetOtherParameterList( t_lib t_cell [?view t_view] [?tool t_tool] [?entry
     s_entry] ) => l_parameterList | nil
```

Description

Returns the `otherParameter` list for the arguments specified. The `otherParameter` list is set with `almSetOtherParameterList`.

Arguments

t_lib	Name of the library in which the cell resides.
t_cell	Name of the cell.
t_view	Name of the view.
t_tool	Name of the tool. This argument is ignored when the *t_view* argument is specified.
s_entry	Parameter entry. The default value is "`otherParameters`".

Value Returned

l_parameterList	The list containing otherParameter names, when defined.
nil	Returns `nil` otherwise.

Example

```
almGetOtherParameterList("analogLib" "vdc"  ?tool "spectre" ?entry
"otherParameters")
```

almGetStringParameterList

```
almGetStringParameterList(t_lib t_cell [?view t_view] [?tool t_tool] [?entry
     s_entry])=> l_parameterList | nil
```

Description

Returns the list of string type parameters for the arguments specified. The parameter list is set with `almSetStringParameterList`.

Arguments

t_lib	Name of the library which the cell resides.
t_cell	Name of the cell.
t_view	Name of the view.
t_tool	Name of the tool. If the *t_view* argument has been specified, and any view-specific information is available for this view, the t_tool argument is ignored.
s_entry	Parameter entry. The default value is `stringParameters`.

Value Returned

`l_parameterList`	The list of parameter names, when defined.
`nil`	Returns nil otherwise.

Example

```
almGetStringParameterList("analogLib" "vdc" ?tool "spectre" ?entry
'stringParameters)
```

almSetStringParameterList

```
almSetStringParameterList(t_lib t_cell l_parameter [?view t_view] [?tool t_tool]
     [?entry s_entry])
```

Description

Sets list of string parameter names for the arguments specified. These parameters are printed to the netlist if the user has given these values.

Arguments

t_lib	Name of the library in which the cell resides.
t_cell	Name of the cell.
l_parameter	List of parameter names. Each element must be a symbol.
t_view	Name of the view.
t_tool	Name of the tool. This argument is ignored when the *t_view* argument is specified.
s_entry	Parameter entry. The default value is stringParameters.

Example

```
almSetStringParameterList( "analogLib" "vpwlf" '(fileName) ?tool "spectre" ?entry
'stringParameters)
```

ancGetSimInstName

```
ancGetSimInstName( l_netlistDpl )
     => t_extName
```

Description

This function provides the external instance name for the current instance being formatted in the netlister. The function must only be used by customized socket netlisters or custom socket netlist procedures. The routine takes into consideration the netlist mode (FNL, HNL or IHNL), case mapping (lower, upper or mixed) and the prefix for the instance.

Argument

l_netlistDpl	Netlist DPL passed to the netlist procedure.

Value Returned

t_extName	Returns the external name for current instance.

Example

```
ancGetSimInstName( netdpl )
```

ancAdjustNameCase

```
ancAdjustNameCase( S_name s_type )
     => S_name
```

Description

The function adjusts case for the name passed, based on the type specified.

Arguments

S_name	The name on which to operate.
s_type	The case adjustment to be made. Valid values are `'lower`, `'upper` or `'mixed`.

Value Returned

`S_name`	Returns the case adjusted name.

Example

```
ancAdjustNameCase( "xI0" 'upper )
```

Returns "XI0".

drbBrowseFormCB

```
drbBrowseFormCB() => t | nil
```

Description

Opens the Browse Project Hierarchy window.

Argument

None

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

msgHelp

```
msgHelp(prodID msgID) => t | nil
```

Description

This function is used to access extended help for an error or a warning message. Currently, Distributed Processing (DP) is the only product that supports this feature.

Arguments

prodID	The product ID. In case of Distributed Processing, it is `'DP`.
msgID	The error message ID. This is the numeric identifier that is available in the error message box.

Value Returned

`t`	If a valid prodID and errorID is passed.
`nil`	If prodID or errorID is not valid.

Example

```
(msgHelp 'DP 8)
```

Displays extended help for the DP error message, whose message ID is 28.

addCheck

```
addCheck(
     t_name
     [?sub          t_sub]
     [?dev          t_dev]
     [?devlist      l_devlist]
     [?prim         t_prim]
     [?mod          t_mod]
     [?instparam    t_instparam]
     [?modelparam   t_modelparam]
     [?opparam      t_opparam]
     [?parameter    t_parameter]
     [?expression   t_expression]
     [?min          t_min]
     [?max          t_max]
     [?regions      t_regions]
     [?duration     t_duration]
     [?message      t_message]
     [?severity     t_severity]
     [?analyses     l_analyses]
     )
```

Description

Adds a new device check. Checks can be added to check a device parameter, a subcircuit parameter, a design variable or an expression. The scope of check is decided by the arguments `?sub`, `?dev`, `?mod`, and `?prim`. The `?min` and `?max` parameters can be used to set up the safe region if the parameter/expression being checked goes out of this region and a violation is reported.

Arguments

t_name	Name for the device check.
t_sub	Subcircuit name. When specified, the check will apply to all instances of this subcircuit.
t_dev	Device name. When specified, the check will apply only to that device. When specified with the subcircuit name, the check will apply to the specified device in all instances of the given subcircuit.

l_devlist	List of device names. When specified, the check will apply only to the those devices. When specified with the subcircuit name, the check will apply to the specified devices in all instances of the given subcircuit.
t_prim	Primitive name. When specified, the check will apply to all instances of this primitive. When specified with the subcircuit name, the check will apply to all instances of this primitive in all instances of the given subcircuit.
t_mod	Model name. When specified, the check will apply to all instances of this model. When specified with the subcircuit name, the check will apply to all instances of this model in all instances of the given subcircuit.
t_instparam	Instance parameter to be checked.
t_modelparam	Model parameter to be checked.
t_opparam	Operating point parameter to be checked.
t_parameter	Circuit parameter (design variable) to be checked.
t_expression	Expression to be checked.
t_min	Lower bound of the safe operating area. When the value of the specified parameter/expression goes below this limit, a violation is reported.
t_max	Upper bound of the safe operating area. When the value of the specified parameter/expression exceeds this limit, a violation is reported.
t_regions	List of safe regions, specified as a space separated string or list of strings. Only valid when *opparam* = `region`. If a device operates outside these regions, a violation is reported.
t_duration	Duration for which a parameter must exceed the bound to cause a violation. Only applies to transient analysis.
t_message	Custom message that is printed when this check is violated.
t_severity	Severity of the violation. Valid Values: `Notice`, `Warning`, `Error`.

l_analyses	List of analyses for which this assert should be turned on. Valid analyses are tran, dc and dcOp.

Value Returned

t_name	Returns the name of device check that was added.
nil	Returns nil if the command is not successful and prints an error.

Examples

```
addCheck("check1" ?primitive "bjt" ?opparam "vbe" ?min "0.7" ?severity "Warning")
```

Checks if vbe for any BJT device falls below 0.7.

```
addCheck("check2" ?primitive "mos3" ?opparam "region" ?regions "triode sat"
?severity "Notice")
```

Checks if any mos3 device goes out of triode or saturation regions.

```
addCheck("check3" ?subckt "amplifier" ?model "mynpn" ?opparam "ibc" ?max 1u
?severity "Warning")
```

Checks if any instance of the model mynpn has base to collector current greater than 1u and limits the check to all instances of the amplifier subcircuit.

deleteChecks

`deleteChecks(`*`t_check1`* `[`*`t_check2 t_check3`*`...])`

Description

Deletes device checks.

Argument

`t_check1 - t_checkn`	Names of checks to be deleted.

Value Returned

`l_checks`	Returns list of checks passed.
`nil`	Returns `nil` and prints an error if the simulator is not Spectre.

Example

```
deleteChecks("check1" "check2" "check3")
```

disableAllChecks

```
disableAllChecks()
```

Description

Disables all device checks.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` and prints an error if the simulator is not Spectre.

Example

```
disableAllChecks()
```

disableChecks

```
disableChecks(
     t_check1 [t_check2 t_check3...]
     )
```

Description

Disables device checks. Only enabled device checks appear in Spectre input deck.

Argument

t_check1 - t_checkn	Names of checks to be disabled.

Value Returned

l_checks	Returns list of checks passed.
nil	Returns `nil` and prints an error if the simulator is not Spectre.

Example

```
disableChecks("check1" "check2" "check3")
```

disableDeviceChecking

```
disableDeviceChecking()
```

Description

Disables device checking. It is the same as turning off the *Enable Device Checking* check box in the *Device Checking Setup* form. No checks are written to Spectre input file. No device checking happens.

Value Returned

`no`	Returns `no` if device checking was already disabled.
`nil`	Returns `nil` if device checking was already disabled.
	Returns `nil` if the command is not successful and prints an error.

Example

```
disableDeviceChecking()
```

displayChecks

```
displayChecks(
     ?resultsDir t_resultsDir [t_check1 t_check2 t_check3...]
     )
```

Description

Displays device checks. If no name is provided, this function displays all checks that have been added so far. If no device checks have been added and a results directory is set by using the *openResults()* command, it prints device checks from the `asserts.info.asserts` file in that results directory. If the results directory is provided, it displays device checks found in the `asserts.info.asserts` file in that results directory.

Arguments

t_resultsDir	Directory containing the PSF files (results). When specified, this argument will only be used internally and will not alter the current results directory which was set by the *openResults()* command.
t_check1 - t_checkn	Names of checks to be displayed.

Value Returned

`nil`	Returns `nil`.

enableAllChecks

```
enableAllChecks()
```

Description

Enables all device checks.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` and prints an error if the simulator is not Spectre.

Example

```
enableAllChecks()
```

enableChecks

```
enableChecks()
```

Description

Enables device checks. Only enabled device checks appear in Spectre input deck.

Argument

t_check1 - t_checkn	Names of checks to be enabled.

Value Returned

l_checks	Returns list of checks passed.
`nil`	Returns `nil` and prints an error if the simulator is not Spectre.

Example

```
enableChecks("check1" "check2" "check3")
```

enableDeviceChecking

```
enableDeviceChecking()
```

Description

Enables device checking. It is the same as turning on the *Enable Device Checking* check box in the *Device Checking Setup* form.

Value Returned

`yes`	Returns `yes` if the command is successful.
`nil`	Returns `nil` if the command is not successful and prints an error.

Example

```
enableDeviceChecking()
```

setDevCheckOptions

```
setDevCheckOptions(
     [?analysis s_analysis]
     [?start t_start]
     [?stop t_stop]
     [?severity t_severity]
     [?enableAll t_enableAll]
     [?disableAll t_disableAll]
     )
```

Description

This function can be used to set various parameters of checklimit statements. User can override the severity specified on individual asserts, provide the time interval during which the checking should be done (applies only to transient analysis), or enable/disable all device checks for the given analysis.

Arguments

s_analysis	Name of analysis.
t_start	Time at which device checking will start in a transient analysis. Not applicable to other analysis.
t_stop	Time at which device checking will stop in a transient analysis. Not applicable to other analysis.
t_severity	If specified, this severity level will override the severity specified for individual asserts. Valid Values: `Warning`, `Notice`, `Error`, `None`.
g_enableAll	If true, all checks are enabled during this analysis.
g_disableAll	If true, all checks are disabled during this analysis.

Value Returned

`nil`	Returns `nil`.

printViolations

```
printViolations(
     ?output t_output
     ?checks l_checks
     ?devices l_devices
     ?models l_models
     ?primitives l_primitives
     ?resultsDir t_resultsDir
     )
```

Description

Prints a summary of all violations. Format of the output is similar to that of the output from the *Print* button on the *Violations Display* form. The `resultsDir` argument can be used to print violations from the different results directory.

Arguments

t_output	Prints the violations to the specified file. If not specified, the violations are printed in the CIW.
l_checks	Prints the violations for the specified checks. If not specified, violations for all the checks are printed.
l_devices	Prints the violations for the specified devices. If not specified, the violations for all the devices are printed.
l_models	Prints the violations for the specified device models. If not specified, the violations for all the device models are printed.
l_primitives	Prints the violations for the specified primitive types or models of the specified primitive type. If not specified, the violations for all the primitives are printed.
t_resultsDir	Specifies the directory containing the PSF files (results). When specified, this argument is used internally and does not alter the current results directory which was set by the *openResults()* command.

Values Returned

`t`	The violations are printed.
`nil`	The violations are not present for the design.

Examples

Example 1

The following example prints all the violations in the CIW.

```
printViolations()
=> Violations for Check1 :
During circuit read in
Instance    Param  Value   Remark
/R1         r      20K     Warning:Exceeded upper limit 10K.
Violations for Check2 :
During circuit read in
Parameter    Value   Remark
tem          27     Warning:Exceeded upper limit 20.
```

Example 2

The following example prints the violations for `Check2` in the CIW.

```
printViolations(?checks "Check2")
=>Violations for Check2 :
During circuit read in
Parameter    Value   Remark
tem          27     Warning:Exceeded upper limit 20.
```

Example 3

The following example prints all the violations to the `printviolation` file.

```
printViolations(?output "./simulation/ampTest/printviolation")
=> t
```

Example 4

The following example prints all the violations to the `printviolation` file with results loaded from the *`./simulation2/ampTest/spectre/schematic`* directory.

```
printViolations(?output "./simulation/ampTest/printviolation" ?resultsDir "./
simulation2/ampTest/spectre/schematic")
=> t
```

Note: You can change the default simulation directory using the `projectDir` environment variable.

captabSummary

```
captabSummary(
     ?resultsDir t_resultsDir
     ?analysis o_analysis
     )
```

Description

Prints a summary of the capacitance table from the specified results directory.

Arguments

o_analysis	Name of the analysis for the capacitance table.
t_resultsDir	Directory containing the PSF files (results).

Example

```
(?resultsDir "./simulation/psf" ?analysis finalTimeCapinfo.captab"))
```

Prints a summary of `finalTimeCapinfo.captab` from `./simulation/psf` directory.

evmOFDM

```
evmOFDM(
     [?waveform1 o_waveform1]
     [?waveform2 o_waveform2]
     [?sigStandard e_sigStandard]
     [?tstart n_tstart
     [?modulationType e_modulationType
     [?fftSize n_fftSize]
     [?PrefixLength n_PrefixLength]
     [?SymbolPeriod n_SymbolPeriod]
     [@key
     [?packet n_packetLength]
     [?modifier e_modifier]
     [?skiplength n_skiplength]
     [?idx n_idx ]
     [?getval g_getval nil]
     )=> constellation o_waveform evm value | nil
```

Description

Processes the I and Q waveform outputs from the hb-envlp simulation run to calculate the Error Vector Magnitude (EVM) and plot the I versus Q scatterplot (constellation). EVM is a useful measurement to describe the overall signal amplitude and phase modulated signal quality. It is based on a statistical error distribution normalized from an ideal digital modulation. Orthogonal Frequency Division Multiplexing (OFDM) is a modern high throughput modulation scheme widely used in wireless signals, such as 802.11a,g,n, where EVM measurement is useful . The EVM is calculated by detecting the I and Q signal levels corresponding to constellation points of each modulation type (can be BPSK, QPSK, 16QAM, and 64QAM) and calculating the difference between the actual signal level and the ideal signal level.

Note: This method is supported for families of waveforms.

Arguments

o_waveform1	The waveform for the I signal.
o_waveform2	The waveform for the Q signal.

e_sigStandard	The standard of the signal. Valid values: 802_11a, 802_11g_ERP, 802_11n_20M_Mix, 802_11n_20M_Legacy, 802_11n_20M_Green, 802_11n_40M_Mix, 802_11n_40M_Legacy, 802_11n_40M_Green, 802_11ac
n_tstart	The start time for the first non-zero point of the signal.
e_modulationType	The modulation type of data carriers.
n_fftSize	The points of FFT, such as 64 for 802.11a.
n_PrefixLength	The length of cyclic prefix, number of samples.
n_SymbolPeriod	The symbol period, such as 4us for long GI 802.11a.
n_packetLength	The length of one packet, can be null if there is only one packet in wave.
e_modifier	The type of showing EVM value, "Percent" or "dB20".
n_skiplength	Used under customer mode, time length between start and data symbol.
n_idx	The index of symbol used for channel estimation.
g_getval	Based on the passed boolean value it returns the following: ■ if nil, it returns EVM value ■ if t, it returns constellation waveform Default is nil.

Values Returned

o_waveform	The waveform object representing the EVM value computed from input waveforms.
Evm Value	The evm value if getVal() is true.
nil	Function call unsuccessful.

Example

```
evmOFDM(real(harmonic(v("/net9" ?result "envlp_fd") '1)) imag(harmonic(v("/net9"
?result "envlp_fd") '1)) "802_11a" 4e-06 'QAM64 64 16 4e-06 ?modifier "Percent"
?packet 0.000173)
```

Calculates the EVM value in term of percent with packet length as 173u and FFT as 64-points.

relxOption

```
relxOption(
     [s_option1 g_optionValue1]....[s_optionN g_optionValueN]
     )=> undefined | nil
```

Description

Enables you to specify the RelXpert options, along with values, to be used by the simulator.

Arguments

s_optionN	Name of the RelXpert option.
g_optionValueN	Value of the RelXpert option.

Value Returned

`undefined`	Undefined.
`nil`	Returns `nil` if the function is not successful.

Example

```
relxOption( 'enableRelxpert  t )
```

Enables `Relxpert`.

asiAddModelLibSelection

```
asiAddModelLibSelection(
     {o_session | o_tool}
     t_modelLibFile
     t_section
     )=> t | nil
```

Description

Adds a model file by adding an entry to the model library file section (corresponding environment variable modelFiles) and calls asiInvalidateControlStmts.

Arguments

o_session	Name of the session.
o_tool	Name of the tool.
t_modelLibFile	Name of the model library file. If this field is blank, no model files are added.
t_section	Name of the model library file section.

Value Returned

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

Example

```
asiAddModelLibSelection (session "/servers/cic_ade/gashish/LAB/lab/Models/
allModels.scs" ""))
```

Adds `allModels.scs` model file and adds an entry to the model library file section.

asiRemoveAllModelLibSelection

```
asiRemoveAllModelLibSelection(
     {o_session | o_tool}
     )=> t
```

Description

Removes all the model files by setting the modelFiles variable to nil. In addition, it also invalidates the asiSendControlStmts flowchart step by calling asiInvalidateControlStmts.

Arguments

o_session	Name of the session.
o_tool	Name of the tool.

Value Returned

t	All model files deleted.

Example

```
asiRemoveAllModelLibSelection (session)
```

Removes all the model files for `session`.

17

OCEAN Script Functions

This chapter describes the functions that let you open an OCEAN script, make changes to it, and save it.

asiOpenOceanScript

```
asiOpenOceanScript(t_fileName)
```

Description

Opens the specified file for writing the OCEAN script. If the file exists, it asks for your permission to overwrite it.

Arguments

fileName	Name of the file or directory to be opened.

Value Returned

fptrReturns the file pointer.

Example

```
asiOpenOceanScript("/usr/mnt3/user/simulation/ckt/spectre/schematic/netlist/
oceanScript")
```

Opens the file `oceanScript`.

asiWriteOceanScript

```
asiWriteOceanScript( p_filePointer o_session ?noRun g_noRun ?fullKey g_fullKey
     ?calledFromCorners g_calledFromCorners)
```

Description

Writes the OCEAN script.

Arguments

p_filePointer	Name of the file pointer, which was opened by calling asiOpenOceanScript.
o_session	Current session.
g_noRun	Flag to determine if the `run()` command and any post-processing plots should be written to the OCEAN script. For example, this flag may be used with a tool that has its own `run()` command.
g_fullKey	Flag to determine if the default values should be written to the OCEAN script.
g_calledFromCorners	Flag to determine whether or not the model file information is to be written to the OCEAN script generated from the Corners tool. By default, it is set to `nil`, so no model file information is written to the OCEAN script generated from the Corners tool.

Values Returned

`t`	Returns `t` if the OCEAN script was written successfully.
`nil`	Returns `nil` if there was an error.

Example

```
saveDefaultsFlag=envGetVal("asimenv.misc" "saveDefaultsToOCEAN" )
asiWriteOceanScript( fp session ?fullKey saveDefaultsFlag )
```

Writes the current session's commands to the OCEAN script if the .cdsenv variable `saveDefaultsFlag` is nil. Otherwise, writes all commands (used as well as unused) to the OCEAN script.

```
asiWriteOceanScript( fp session)
```

Writes the current session's commands to the OCEAN script.

```
asiWriteOceanScript( fp session ?noRun t)
```

Writes the OCEAN script without adding `run()` and plots to the end.

asiCloseOceanScript

```
asiCloseOceanScript( p_filePointer )
```

Description

Closes the OCEAN script

Arguments

p_filePointer	File pointer to the OCEAN script file.

Value Returned

`t`	Returns `t` if successful.
`nil`	Returns `nil` otherwise.

Example

```
asiCloseOceanScript(fptr)
```

Closes the specified file.

18

Waveform Data Objects

Waveform data objects are used in simulation results manipulation and display in the Virtuoso analog design environment. In this chapter, functions are described that you can use to create, modify, and analyze these objects.

Waveform data objects are represented by the special type *srrWaveID*. Each ID uniquely identifies a waveform data object. A waveform data object has two data-vectors, the X and the Y data vector. Similar to waveform data objects, data vectors are represented by the special type *srrVecID*. Each ID uniquely identifies a data vector.

Note: SKILL functions, such as *declare()* and *makeVector()*, return a vector object, which is conceptually similar to but structurally different from the vector returned by public APIs, such as *drCreateVec()*. The former is a (SKILL) language-supported data structure. The latter is an application-supported data structure, for example, the data vector used in functions. *srrWaveXXXX* public APIs require the application-supported vector type, not the skill vector type or array.

Data Values

All the values in a data vector must be of the same data type. However, the X and Y vectors in a waveform data object can be of different data types.The data types and their symbolic representation that are supported for Data Vectors in SKILL are:

- 'intlong for integer numbers
- 'double for double-precision floating-point numbers
- 'doublecomplex for complex numbers
- 'string for strings

When you create a new waveform data object, you must first create the X and Y vectors with *drCreateVec*. You can use *drCreateWaveform* with the X and the Y vectors as arguments to create a waveform.

drAddElem

```
drAddElem(o_vec g_value) => t | nil
```

Description

Puts *g_value* after the last element in the data vector *o_vec*. *g_value* must have the same data type as the data vector.

Arguments

o_vec	Data vector to which a new element *g_value* will be appended.
g_value	New element to be added at end of data vector *o_vec*.

Value Returned

t	Returns t if element is added successfully.
nil	Returns nil if there is an error.

Examples

```
vec = drCreateVec('double 5)
drAddElem(vec 10.0)
```

drGetElem

```
drGetElem( o_vec x_index) => g_result
```

Description

Returns the *x_index*th element of the data vector *drVecID*, assuming a zero- based index.

Arguments

o_vec	Target data vector.
x_index	Target element of data vector *drVecID*.

Value Returned

g_result	Returns the contents of element *x_index* of data vector *o_vec*.

Example

```
vec = drCreateVec( 'double '(1 2 3 4 5))
drGetElem(vec 2)
```

drSetElem

```
drSetElem(o_vec x_index g_value) => t | nil
```

Description

Replaces the *x_index*th element of the data vector *o_vec* with *g_value*. *g_value* must have the same data type as the data vector.

Arguments

o_vec	Data vector to which a new element *g_value* will be appended.
x_index	The index for which the value is replaced
g_value	New element to be added at end of data vector *o_vec*.

Value Returned

`t`	Returns `t` if the element is replaced.
`nil`	Returns `nil` if there is an error.

Example

```
vec = drCreateVec('double 5)
```

drSetElem(*vec 0 10.0)*

drCreateVec

```
drCreateVec( s_dataType x_length ) => o_vec/nil
drCreateVec( s_dataType l_values ) => o_vec/nil
```

Description

Creates a new data vector.

Arguments

s_dataType	Data type. Possible values are described in Data Values on page 675.
x_length	Length of the vector. The value of this argument must be greater than 0.
l_values	List of values.

Value Returned

o_vec	Returns the created vector if successful.
`nil`	Returns `nil` if there is an error.

Example

```
vec = drCreateVec('double 5)
```

drCreateEmptyWaveform

```
drCreateEmptyWaveform() => o_waveform
```

Description

Creates an empty waveform data object.

Arguments

None.

Value Returned

o_waveform	Returns the waveform object created.

Example

```
drCreateEmptyWaveform()
```

drCreateWaveform

```
drCreateWaveform( o_xvec o_yvec ) => o_wave
```

Description

Creates a waveform data object with the vectors specified.

Arguments

o_xvec	X vector.
o_yvec	Y vector.

Value Returned

o_wave	Returns the created waveform if successful.
`nil`	Returns `nil` if there is an error.

Example

```
wave = drCreateWaveform( drCreateVec( 'double 5 ) drCreateVec( 'double 5 ) )
```

drGetWaveformXType

```
drGetWaveformXType( o_wave ) => s_dataType
```

Description

Returns the X vector data type of the waveform data object *o_wave*.

Arguments

o_wave	Waveform data object.

Value Returned

s_dataType	Returns the symbol indicating X vector data type as described Data Values on page 675.

Example

```
wave = drCreateWaveform( drCreateVec( 'double 5 ) drCreateVec( 'double 5 ) )
drGetWaveformXType( wave )
```

drGetWaveformXVec

```
drGetWaveformXVec( o_wave ) => o_vec
```

Description

Returns the X vector object ID of the waveform data object *o_wave*.

Arguments

o_wave	Waveform data object.

Value Returned

o_vec	Returns the X vector object ID.

Example

```
wave = drCreateWaveform( drCreateVec( 'double 5 ) drCreateVec( 'double 5 ) )
drGetWaveformXVec( wave )
```

drGetWaveformYType

```
drGetWaveformYType( o_wave ) => s_dataType
```

Description

Returns the Y vector data type of the waveform data object *o_wave*.

Arguments

o_wave	Waveform data object.

Value Returned

s_dataType	Returns the symbol indicating Y vector data type as described in Data Values on page 675.

Example

```
wave = drCreateWaveform( drCreateVec( 'double 5 ) drCreateVec( 'double 5 ) )
drGetWaveformYType( wave )
```

drGetWaveformYVec

```
drGetWaveformYVec( o_wave ) => o_vec
```

Description

Returns the Y vector object ID of the waveform data object *o_wave*.

Arguments

o_wave	Waveform data object.

Value Returned

o_vec	Returns the Y vector object ID.

Example

```
wave = drCreateWaveform( drCreateVec( 'double 5 ) drCreateVec( 'double 5 ) )
drGetWaveformYVec( wave )
```

drPutWaveformXVec

```
drPutWaveformXVec( o_wave o_vec ) => t | nil
```

Description

Puts the data vector *o_vec* into the waveform data object *o_wave*. *o_vec* is the X vector of the waveform data object. If the *o_wave* already contains a Y vector, the length of *o_vec* must be the same as that of the Y vector.

Arguments

o_wave	Waveform data object.
o_vec	The new X data vector.

Value Returned

t	Returns t if data vector is added successfully.
nil	Returns nil if there is an error.

Example

```
wave = drCreateEmptyWaveform()
vec = drCreateVec( 'double '(1 2 3 4 5))
drPutWaveformXVec( wave vec )
```

drPutWaveformYVec

```
drPutWaveformYVec( o_wave o_vec ) => t | nil
```

Description

Puts the data vector *o_vec* into the waveform data object *o_wave*. *o_vec* is the Y vector of the waveform data object. If the *o_wave* already contains a X vector, the length of *o_vec* must be the same as that of the X vector.

Arguments

o_wave	Waveform data object.
o_vec	The new Y data vector.

Value Returned

t	Returns t if data vector is added successfully.
nil	Returns nil if there is an error.

Example

wave = drCreateEmptyWaveform()

```
vec = drCreateVec( 'double '(1 2 3 4 5))
drPutWaveformYVec( wave vec )
```

drIsDataVector

```
drIsDataVector( g_value ) => t | nil
```

Description

Returns t when *g_value* is a valid drVector data object and `nil` if it is not one.

Arguments

g_value	Value checked.

Value Returned

`t`	Returns `t` when *g_value* is a drVector data object.
`nil`	Returns `nil` if *g_value* is not a drVector data object.

Example

```
wave = drCreateEmptyWaveform()
drIsDataVector( wave ) => nil
```

drIsParamWave

```
drIsParamWave( g_value ) => t | nil
```

Description

Returns t when *g_value* is a family, for example a parametric wave.

Arguments

g_value	Value checked.

Value Returned

t	Returns t when *g_value* is a family.
nil	Returns nil if *g_value* is not a family.

Example

After running a Parametric Analysis, evaluate any net expression, such as:

```
fam = VT("/out")
drIsParamWave( fam ) => t

wave = drCreateEmptyWaveform()
drIsParamWave( wave ) => nil
```

drIsWaveform

```
drIsWaveform( g_value ) => t | nil
```

Description

Returns t when *g_value* is a waveform data object and `nil` if it is a family.

Arguments

g_value	Value checked.

Value Returned

`t`	Returns `t` when *g_value* is a waveform data object.
`nil`	Returns `nil` if *g_value* is not a waveform data object.

Example

```
wave = drCreateEmptyWaveform()
drIsWaveform( wave ) => nil
```

drType

```
drType( o_vec ) => s_type
```

Description

Returns the data type of the data vector *o_vec*.

Possible values are described Data Values on page 675.

Arguments

o_vec	The vector for which the data type is returned.

Value Returned

s_type	Returns a symbolic representation of the type of the data vector. The values are described Data Values on page 675.

Example

```
vec = drCreateVec( 'double '(1 2 3 4 5))
drType( vec )
```

drVectorLength

```
drVectorLength( o_vec ) => x_length
```

Description

Returns the length of the data vector *o_vec*.

Arguments

o_vec	Target data vector.

Value Returned

x_length	Returns the length of data vector *o_vec*.

Examples

```
vec = drCreateVec( 'double '(1 2 3 4 5))
drVectorLength( vec )
```

famAddValue

```
famAddValue( o_family g_sweepValue g_value ) => o_family
```

Description

Adds a waveform associated with *sweepValue* to a family if values are specified for both *sweepValue* and *value*.

Arguments

o_family	The family to receive the new value.
g_sweepValue	The sweep value with which to associate the new waveform.
g_value	The new waveform to add.

Value Returned

o_family	The modified family.

Examples

```
fam = famCreateFamily( "prf" 'double )
srrWave:17518616
famAddValue( fam -15 wave15 )
srrWave:17518616
famAddValue( fam -10 wave10 )
srrWave:17518616
famAddValue( fam -5 wave5 )
srrWave:17518616
```

famCreateFamily

```
famCreateFamily( s_sweepName s_varType ) => o_family
```

Description

Creates a empty data structure called a family. When filled with data, it is suitable to be plotted plotted as a family of curves. Each waveform has a name given by *s_varName* and the values are added with famAddValue.

Arguments

s_sweepName	The string name of the sweep variable.
s_varType	One of these variable types: `'double`, `'doublecomplex`, `'float`, `'intlong`, `'signal`, `'string`. The *varType* of `'float` is equivalent to `'double` and `'signal` is equivalent to `'string`.

Value Returned

o_family	An empty family with the variable label `s_var`.

Examples

```
fam = famCreateFamily( "prf" 'double )
srrWave:17518616

famAddValue( fam -15 wave15 )
srrWave:17518616

famAddValue( fam -10 wave10 )
srrWave:17518616

famAddValue( fam -5 wave5 )
srrWave:17518616
```

famGetSweepName

```
famGetSweepName( o_family [ x_dim ] ) => s_sweepName
```

Description

Returns the name of the sweep variable of the parametric waveform and the dimension supplied.

Arguments

o_family	A family name.
x_dim	An integer that specifies how deep the function should go before returning the sweep name.

Value Returned

s_sweepName	The string name of the sweep variable.

Examples

```
fam1 = famCreateFamily( "prf" 'double )
srrWave:29335584
fam2 = famCreateFamily( "prf" 'double )
srrWave:29335592
fam3 = famCreateFamily( "ofreq" 'double )
srrWave:29335600
famAddValue( fam3 999M fam1)
srrWave:29335600
famAddValue( fam3 1G fam2)
srrWave:29335600
famGetSweepName( fam3 )
"ofreq"
famGetSweepName( fam3 0 )
"ofreq"
famGetSweepName( fam3 1 )
"prf"
```

famGetSweepValues

```
famGetSweepValues( o_family ) => l_values
```

Description

Returns the values of the sweep variable of the family specified. The returned list is sorted in increasing order.

Arguments

o_family	A family name.

Value Returned

l_values	A list of values of the sweep variable.

Examples

```
foreach( v famGetSweepValues( fam )
printf( "%L\n" v))
```

famIsFamily

```
famIsFamily( g_arg ) => t | nil
```

Description

Checks whether the argument specified is a family with at least one waveform.

Arguments

g_arg	Any expression.

Value Returned

`t`	Returns `t` when *g_arg* is a family.
`nil`	Returns `nil` if *g_arg* is not a family.

Examples

```
famIsFamily( 3 )
nil

fam1 = famCreateFamily( "prf" 'double )
srrWave:50152744

famIsFamily( fam1 )
nil

famAddValue( fam1 -10 drCreateEmptyWaveform() )
srrWave:50152744

famIsFamily( fam1 )
t
```

famMap

famMap(*s_func o_family* [*args ...*]) => *o_result*

Description

Applies a function with a set of arguments to each member of a family of waveforms.

Arguments

s_func	A function specified as a string or symbol.
o_family	A family.

Value Returned

o_result	A family with a structure similar to the input family.

Examples

```
;; After running parametric pnoise analysis
wave1 = getData( "out" ?result 'pnoise )
srrWave:50152592
ocnPrint( wave1 )

freq (Hz)       quotient (getD      quotient (getD
prf             -10                 -9

901M            2.39482n            2.6192n
902M            2.04681n            2.1431n
903M            1.84256n            1.89187n
904M            1.75846n            1.78815n
905M            1.71612n            1.73575n
t
ocnPrint( famMap( 'quotient wave1 2.0 ))

freq (Hz)       quotient (getD      quotient (getD
prf             -10                 -9

901M            1.19741n            1.3096n
```

```
902M            1.02341n            1.0715n
903M            0.92128n            0.94594n
904M            0.87923n            0.89408n
905M            0.85806n            0.86788n
t
;; This is equivalent to ocnPrint( wave1 | 2.0 )
```

famValue

```
famValue( o_family g_sweepValue ) => o_waveformOrFamily
```

Description

Returns the waveform whose *sweepName* has the value specified using *sweepValue*.

Arguments

o_family	A family of waveforms.
o_sweepValue	The value of sweep to retrieve. This must be an exact match with the value specified using famAddValue.

Value Returned

o_waveformOrFamily	The waveform or family whose sweep value was specified in the argument.

Examples

```
wave1
srrWave:29298888

wave2
srrWave:29298896

famStr = famCreateFamily( "name" 'string )
srrWave:29298920

famAddValue( famStr "one" wave1 )
srrWave:29298920

famAddValue( famStr "two" wave2 )
srrWave:29298920

famValue( famStr "one")
srrWave:29298968
;; although it is a differen waveform it is equivalent.

equal( wave1 famValue( famStr "one" ))
t

famValue( famStr "three" )
nil


famNum = famCreateFamily( "prf" 'double )
srrWave:29299008

famAddValue(famNum 1.0 wave1)
srrWave:29299008
```

```
famAddValue( famNum 2.0 wave2 )
srrWave:29299008

famValue( famNum 1.5 )
srrWave:29299056   ;; interpolated waveform is returned
```

19

Mixed Signal Simulation Functions

This chapter contains the following sections:

Simulation Functions for Direct Interfaces

asiVerilogNetlistMoreCB

```
asiVerilogNetlistMoreCB()
     => t
```

Description

Displays the Verilog HNL Netlisting Option form.

Arguments

None.

Value Returned

`t`	Always returns t.wsd

asiGetDigitalNetlistFileName

```
asiGetDigitalNetlistFileName( o_session )
     => t_digitalNetlistFileName
```

Description

Returns the digital netlist file name.

Arguments

o_session	Simulation session object.

Value Returned

t_digitalNetlistFileName Returns the digital netlist file name.

asiConstructDigitalNetlist

```
asiConstructDigitalNetlist( o_session )
     => t | nil
```

Description

Constructs the digital netlist file for viewing.

This file is a concatenated form of the following files :

- The template stimulus file (`testfixture.template`)
- The verimix stimulus file (`testfixture.verimix`)
- The verimix specific defines (`simOptions.verimix`)
- The part of the netlist file that defines Interface Elements and Verimix Synchronization task statements (`IE.verimix`)
- Parasitic simulation annotate definitions (annotate_msb)
- The `$save_waveform` definitions (saveDefs)
- The hierarchical netlist blocks concatenated into one file (netlist).

Arguments

o_session	Simulation session object.

Value Returned

`t`	Returns `t` if the netlist file was created.
`nil`	Returns `nil` if there was an error.

asiInitializeNetlisterMixed

```
asiInitializeNetlisterMixed( o_session )
     => t_mixedSignalDesignObject | nil
```

Description

Initializes the mixed signal netlister. Partitions the design and then calls `nlCreateDesign` to get the design object. Does not re-partition if the design has not changed since last partition.

Arguments

o_session	The OASIS session object.

Value Returned

t_mixedSignalDesignObject	Returns the object representing the mixed signal design if the netlister is initialized.
`nil`	Returns `nil` if there was an error.

asiNetlistMixed

```
asiNetlistMixed( o_session )
     => g_status | nil
```

Description

This method performs mixed netlisting. It creates a formatter object with `nlCreateFormatter`, after which the netlister is run and a netlist is generated with `nlNetlist`.

This routine is called by asiNetlist . You should not make a direct call to this routine.

Arguments

o_session	Simulation session object.

Value Returned

g_status	Returns a DPL of file names created by the netlister if successful.
`nil`	Returns `nil` if there was an error.

asiGetVerilogCommandLineOption

```
asiGetVerilogCommandLineOption( o_session )
     => t_verilogCommandLineOption
```

Description

Returns the verilog simulator command line options

Arguments

o_session	Simulation session object.

Value Returned

t_verilogCommandLineOption	Retuens the command line options for the verilog simulator with mixed mode capability.

asiGetDigitalCommandLineOption

```
asiGetDigitalCommandLineOption( o_session )
     => t_digitalCommandLineOption
```

Description

Returns the digital part of the mixed simulation command line options.

Arguments

o_session	Simulation session object.

Value Returned

t_digitalCommandLineOption	Returns the digital part of the mixed simulation command line options.

asiPrepareDigitalSimulation

```
asiPrepareDigitalSimulation( o_session )
     => t | nil
```

Description

Performs digital run directory clean up. It removes the file that stores the exit code for any previous digital simulation and cleans up the digital simulator old log file.

Arguments

o_session	Simulation session object.

Value Returned

t	Returns t if the old log files are deleted.
nil	Returns nil if there was an error.

asiCheckDigitalSimulationSuccess

```
asiCheckDigitalSimulationSuccess( o_session )
     => t | nil
```

Description

Reports the success or failure of the simulation by looking at the digital simulator status file. Further, if no digital signals are saved by the user, this routine updates the `logFileVerilog` file to indicate that no digital waveforms are available for viewing.

Arguments

o_session	Simulation session object.

Value Returned

`t`	Returns `t` if the simulation was successful.
`nil`	Returns `nil` if there was an error.

Simulation Functions for Direct and Socket Interfaces

asiGetNetworkId

```
asiGetNetworkId( o_session )
     => o_networkId
```

Description

Returns the network ID of a mixed signal design.

Arguments

o_session	Simulation session object.

Value Returned

o_networkId	The network Id of the mixed signal design

asiGetDigitalStimulusFileName

```
asiGetDigitalStimulusFileName( o_session )
     => t_digitalStimulusFileName
```

Description

Returns the digital stimulus file name. This file name is the concatenated path string of the digital netlist run directory followed by `testfixture.verimix`.

Arguments

o_session	Simulation session object.

Value Returned

t_digitalStimulusFileName Returns the full path of the digital stimulus file name.

asiEditDigitalStimulus

```
asiEditDigitalStimulus( o_session )
     => o_childId | nil
```

Description

This method edits the digital stimulus file. It gets the full path to the stimulus file by calling `asiGetDigitalStimulusFileName`. If the stimulus file does not exist, the `asiNetlist` flowchart step is executed to run the netlister and create a digital stimulus file. Finally the stimulus file is opened in an editor.

Arguments

o_session	The OASIS session object.

Value Returned

o_childId	Returns the handle on the child editor process.
`nil`	Returns `nil` if there was an error.

Example

```
asiInitializeNetlisterMixed( session )
```

asiPartitionDesign

```
asiPartitionDesign( o_session )
     => t | nil
```

Description

This method partitions a mixed signal design into analog and digital parts.

Arguments

o_session	Simulation session object.

Value Returned

`t`	Returns `t` if the design is successfully partitioned.
`nil`	Returns `nil` if there was an error.

Example

```
asiNetlistMixed( session )
```

asiGetDigitalSimulatorLogFileName

```
asiGetDigitalSimulatorLogFileName( o_session )
     => t_verilogLogFileName
```

Description

design.

Arguments

o_session	Simulation session object.

Value Returned

t_verilogLogFileName Returns the digital simulator log file name.

asiGetDigitalSimExecName

```
asiGetDigitalSimExecName( o_session )
     => t_digitalSimExecName
```

Description

Displays the verimix executable name.

Arguments

o_session	Simulation session object.

Value Returned

t_digitalSimExecName Returns the name of the digital simulator executable.

asiSetVerilogHost

```
asiSetVerilogHost( o_session t_host )
     => t_digitalSimulatorHostName
```

Description

Sets the digital simulator host name for mixed signal simulation.

Arguments

o_session	Simulation session object.
t_host	Name of the host on which you want to run the digital simulator.

Value Returned

t_digitalSimulatorHostName	Returns the name of the host on which you want to run the digital simulator.

asiSetVerilogHostMode

```
asiSetVerilogHostMode( o_session t_hostMode )
     => t_digitalSimulatorHostMode
```

Description

Specifies whether the digital simulator will run locally or on a remote host. If you specify `remote`, you must specify the host name by using the `asiSetVerilogHost` command.

Arguments

`o_session`	Simulation session object.
`t_hostMode`	Should be specified as either `local` or `remote`.

Value Returned

`t_digitalSimulatorHostMode` Returns the digital Simulator host mode for mixed signal simulation.

asiGetVerilogHost

```
asiGetVerilogHost( o_session )
     => t_digitalSimulatorHostName
```

Description

Returns the digital simulator host name for mixed signal simulation. This method first verifies if host name was set using *asiSetVerilogHost*. If yes, it returns that host name. Otherwise, it returns the default value of digitalHost in the `.cdsenv` file for tool asimenv and partition startup.

Arguments

o_session Simulation session object.

Value Returned

t_digitalSimulatorHostName Name of the host on which the digital simulator will be run.

asiGetVerilogHostMode

```
asiGetVerilogHostMode( o_session )
     => t_digitalSimulatorHostMode
```

Description

Returns the digital simulator host mode for mixed signal simulation. This method first verifies if host mode was set using `asiSetVerilogHostMode`. If yes, it returns that mode. Otherwise, it returns the default value of `digitalHostMode` in the `.cdsenv` file for tool asimenv and partition startup.

Arguments

`o_session`	Simulation session object.

Value Returned

`t_digitalSimulatorHostMode`	Returns the digital Simulator host mode for mixed signal simulation.

asiGetAnalogRunDir

```
asiGetAnalogRunDir( o_session )
     => t_AnalogSimulatorRunDir
```

Description

Returns the analog simulator run directory.

Arguments

o_session	Simulation session object.

Value Returned

t_AnalogSimulatorRunDir Analog simulator run directory.

asiGetDigitalRunDir

```
asiGetDigitalRunDir( o_session )
     => t_DigitalSimulatorRunDir
```

Description

Returns the digital simulator run directory.

Arguments

o_session	Simulation session object.

Value Returned

t_AnalogSimulatorRunDir Returns the digital simulator run directory.

asiGetAnalogKeepList

```
asiGetAnalogKeepList( o_session )
     => l_AnalogSignalDescriptionList
```

Description

Returns the list of user-selected output signals that are analog. This method first checks if the current simulation session is mixed signal. If yes, then it returns all the user-selected output signals within the analog partition of the mixed signal design. Otherwise, it calls `asiGetKeepList` to get the list of signals and currents to be saved during simulation.

Arguments

o_session	Simulation session object.

Value Returned

l_AnalogSignalDescriptionList	Returns the list of analog signals to be saved during simulation.
`nil`	Returns `nil` if no signals were specified to be kept during simulation.

asiGetDigitalKeepList

```
asiGetDigitalKeepList( o_session )
     => l_DigitalSignalDescriptionList
```

Description

Returns the list of user-selected output signals that are digital.

Arguments

o_session	Simulation session object.

Value Returned

l_DigitalSignalDescriptionList	Returns the list of digital signals to be saved during simulation.
nil	Returns nil if no signals were specified to be kept during simulation.

asiInitMixedKeepOption

```
asiInitMixedKeepOption( o_tool ) => t
```

Description

Initializes the mixed signal keep options variables.

Arguments

o_tool	Tool

Return Value

t	Success

Example

```
asiInitMixedKeepOption(asiGetTool('spectreSVerilog))
```

asiInitVerilog

```
asiInitVerilog( o_tool ) => t
```

Description

Initializes the verilog tool.

Arguments

o_tool	Tool

Return Value

t	Success

Example

```
asiInitVerilog(asiGetTool('spectreVerilog))
```

asiInitVerilogEnvOption

```
asiInitVerilogEnvOption( o_tool ) => t
```

Description

Initializes the base Verilog environment options.

Arguments

o_tool	Tool

Return Value

t	Success.

Example

```
asiInitVerilogEnvOption(asiGetTool('spectreSVerilog))
```

asiInitVerilogFNLEnvOption

```
asiInitVerilogFNLEnvOption( o_tool ) => t
```

Description

Initializes the Verilog FNL netlisting options.

Arguments

o_tool	Tool

Return Value

t	Success.

Example

```
asiInitVerilogFNLEnvOption(asiGetTool('spectreVerilog))
```

asiInitVerilogHNLEnvOption

```
asiInitVerilogHNLEnvOption( o_tool ) => t
```

Description

Initializes the Verilog HNL netlisting options.

Arguments

o_tool	Tool

Return Value

t	Success.

Example

```
asiInitVerilogHNLEnvOption(asiGetTool('spectreSVerilog))
```

asiInitVerilogSimOption

```
asiInitVerilogSimOption( o_tool ) => t
```

Description

Initializes the simulation options for verilog.

Arguments

o_tool	Tool

Return Value

t	Success.

Example

```
asiInitVerilogSimOption(asiGetTool('spectreSVerilog))
```

asiSetAnalogSimulator

```
asiSetAnalogSimulator( o_tool s_analogSimulator ) => s_analogSimulator
```

Description

Sets the analog simulator.

Arguments

o_tool	Tool
s_analogSimulator	Analog simulator name

Return Value

s_analogSimulator	analog simulator name that has been set

Example

```
asiSetAnalogSimulator( asiGetTool('spectreSVerilog) 'spectre )
```

asiSetDigitalSimulator

```
asiSetDigitalSimulator( o_tool s_digitalSimulator) => s_digitalSimulator
```

Description

Sets the digital simulator.

Arguments

o_tool	Tool
s_digitalSimulator	Digital simulator name

Return Value

s_digitalSimulator	digital simulator name that has been set

Example

```
asiSetDigitalSimulator( asiGetTool('spectreSVerilog) 'verilog )
```

mspDisplaySetPartSetupForm

```
mspDisplaySetPartSetupForm() => t | nil
```

Description

This function displays the Partitions Options form, which can be used to edit and set the values for the Analog Stop View Set and Digital Stop View Set.

Arguments

None

Return Value

`t`	Success
`nil`	Failure

Example

```
mspDisplaySetPartSetupForm()
```

mspEditIEProps

```
mspEditIEProps( t_objectType ) => t | nil
```

Description

This is the main entry procedure for the Interface Element property editor. It takes an object type. The object type can be one of instTerm, instance, terminal, cellView, lib or default. This function first calls a selection function appropriate for the object type which prompts the user to select an object. When the object has been selected, the selection function will execute its callback. An exception to this is that for lib and default objectType values, no selection functions are called. Next, the IE model property editor is called to create a parameter entry form for the current ieModel. The callbacks to this form will attach properties to the object selected earlier.

Arguments

t_objectType	object type

Return Value

t	Success

Note: This value is returned immediately after the function is called denoting that the function has completed execution. The function does not wait for the actions mentioned in the description to complete. It is only responsible for invoking the selection function which generates the callback action and subsequent updation of properties is needed.

Example

```
mspEditIEProps("cellView")
```

Functions for Formatting Hierarchical Interface Elements

Formatting for pmos, rpmos, nmos, or rnmos Gates

It is not necessary to write these functions unless the default definition suffices.

hnlVerilogPrintNmosPmos

```
hnlVerilogPrintNmosPmos( t_name ) -> t
```

Description

Used by netlister to print `pmos`, `rpmos`, `nmos`, or `rnmos` gates.

Arguments

t_name	The name of the gate. Valid values are `pmos rpmos nmos` and `rnmos`.

Value Returned

`t`

Formatting for cmos and rcmos Gates

It is not necessary to write these functions unless the default definition suffices.

hnlVerilogPrintCmos

```
hnlVerilogPrintCmos( t_name ) -> t
```

Description

Used by netlister to print `cmos`, `rcmos` gates.

Arguments

t_name	The name of the gate. Valid values are `cmos`, `rcmos`.

Value Returned

`t`

nlGetCdf

```
nlGetCdf(o_inst) => cdfId
```

Description

Obtains the CDF information of the IE cell. CDF information contains the default IE parameter, default IE macro model file name and parameter mapping from CDF to simulator specific names.

Arguments

o_inst	The IE object (nlIEInstance)

Values Returned

o_cdfId	CDF data of the IE cell

Example

```
spectreSimInfo = nlGetCdf(ie_inst)->simInfo->spectre
```

nlGetCellName

```
nlGetCellName(o_inst) => t_cellName
```

Description

Gets the CDBA cell name of the IE instance. This cell name is the name of the IE cell which is used as the basis for modeling the IE. Examples of cell names are `MOS_a2d` and `TTL_d2a`.

Arguments

o_inst	The hierarchical IE object (`nlHierIEInstance`)

Values Returned

`t_cellName`	The cell name of the IE given as a string

Example

```
printf("Cell name of IE is %s.\n" nlGetCellName(inst))
```

nlGetLibName

```
nlGetLibName(o_inst) => t_libName
```

Description

Gets the CDBA library name of the IE instance. This library is the library containing the IE cell which is used as the basis for modeling the IE. Examples of library names are `analogLib` and `ieLib`.

Argument

o_inst	The hierarchical IE object (`nlHierIEInstance`)

Return

`t_libName`	The library name of the IE given as a string.

Example

```
printf("Library name of IE is %s.\n" nlGetLibName(inst))
```

20

Sev Functions

The sev functions are used by the Virtuoso Analog Simulation Environment to build and maintain the information and interfaces used by the environment. Most of these functions require the `session` argument. You can obtain this argument by using the `sevSession`() function or by calling these functions from a `.menus` file.

sevSetMainWindowPulldownMenus

```
sevSetMainWindowPulldownMenus(menus)
     => t | nil
```

Description

Sets the menus for the Virtuoso Analog Simulation Environment window.

Arguments

menus	The list of menu names

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevSetMTSMode

```
sevSetMTSMode(o_session)
     => t | nil
```

Description

Returns the status of the MTS mode.

Arguments

o_session	The simui session object

Value Returned

`t`	Enables the MTS mode.
`nil`	Disables the MTS mode.

sevMTSMode

```
sevMTSMode(o_session)
     => t | nil
```

Description

Returns the status of the MTS mode.

Arguments

o_session	The simui session object

Value Returned

`t`	Enables the MTS mode.
`nil`	Disables the MTS mode.

sevMTSOptions

```
sevMTSOptions(o_session)
     => t | nil
```

Description

Invokes the MTS option form if MTS mode is enabled. A dialog box appears if the MTS option is disabled.

Arguments

o_session	The simui session object

Value Returned

`t`	Displays the MTS options form.
`nil`	Does not display the MTS options form.

sevSetSchematicPulldownMenus

```
sevSetSchematicPulldownMenus(menus)
     => t | nil
```

Description

Sets the simulation menus on the schematic window.

Arguments

menus	The list of menu entries

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevSetTypeInWindowPulldownMenus

```
sevSetTypeInWindowPulldownMenus(menus)
     => t | nil
```

Description

Sets the menu on the simulator type in window.

Arguments

menus	The list of menu entries

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevSetMenuItemLists

```
sevSetMenuItemLists(lists)
     => t | nil
```

Description

Creates the menu item lists.

Arguments

lists	The list of menu entries

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevAddMenuItemLists

```
sevAddMenuItemLists(lists)
     => t | nil
```

Description

Adds menu items to an existing menu item list.

Arguments

lists	The list of menu entries

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevDirectPlotMenu

```
sevDirectPlotMenu(session items)
     => t | nil
```

Description

Creates the direct plot menu item list.

Arguments

session	The simulation environment session.
items	The list of menu entries.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevEnvironment

```
sevEnvironment(session)
     => o_session | nil
```

Description

Displays the Oasis session object tied to the simulation environment session.

Arguments

session	The simulation environment session.

Value Returned

o_session	Returns the Oasis session object
nil	Returns nil if there is no Oasis session object currently tied to the simulation environment.

sevNoEnvironment

```
sevNoEnvironment(session)
     => t | nil
```

Description

Indicates whether an Oasis environment is tied to the simulation environment.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevSaveState

```
sevSaveState(session)
     => t | nil
```

Description

Displays the *Saving State* form that lets you save the current state of the simulation environment.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevLoadState

```
sevLoadState(session)
     => t | nil
```

Description

Displays the *Loading State* form that lets you load a saved state into the current simulation environment.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevSaveOceanScript

```
sevSaveOceanScript(session)
     => t | nil
```

Description

Displays the Save Ocean Script to File form that lets you save an Ocean script that will regenerate the current session to the specified file.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevEditOptions

```
sevEditOptions(session)
     => t | nil
```

Description

Displays the Editing Session Options form that lets you edit the options for the given simulation environment session.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevOpenSchematic

```
sevOpenSchematic(session)
     => t | nil
```

Description

Displays a schematic window for the design that is tied to the simulation environment.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevMenuItems

```
sevMenuItems(session name)
     => t | nil
```

Description

Displays the menu item list that corresponds to the named menu item.

Arguments

session	The simulation environment session.
name	The name of menu item from the menu item list.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevReset

```
sevReset(session)
     => t | nil
```

Description

Resets the simulation environment session to its default values.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevQuit

```
sevQuit(session)
     => t | nil
```

Description

Quits the simulation session.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevCreateMainWindow

```
sevCreateMainWindow(session)
     => t | nil
```

Description

Creates the main simulation environment window.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevChooseSimulator

```
sevChooseSimulator(session)
     => t | nil
```

Description

Displays the Choosing Simulator/Directory/Host form that lets you choose the simulator you want to use, the run directory, and the host machine.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevChooseTemperature

```
sevChooseTemperature(session)
     => t | nil
```

Description

Displays the Setting Temperature form that lets you set the simulation temperature.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevMpuTool

```
sevMpuTool(session)
     => t | nil
```

Description

Displays the Setting Model Path form that lets you select the paths to the model files.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevChooseEnvironmentOptions

```
sevChooseEnvironmentOptions(session)
     => t | nil
```

Description

Displays the Environment Options form that lets you select the environment options for the simulation environment session.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevEditStimulus

```
sevEditStimulus(session type)
     => t | nil
```

Description

Displays the specified stimulus in a window for editing.

Arguments

session	The simulation environment session.
type	The type of stimulus - `analog` or `digital`.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevNonMixedSignal

```
sevNonMixedSignal(session)
     => t | nil
```

Description

Indicates whether the session is running a mixed signal simulation.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the session is running mixed signal simulation.
`nil`	Returns `nil` if the session is running analog simulation.

sevEditSimulationFile

```
sevEditSimulationFile(session type)
     => t | nil
```

Description

Displays the specified file in a window for editing.

Arguments

session	The simulation environment session.
type	The type of simulation file to edit - `include`, `model`, or `stimulus`.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevChooseDesign

```
sevChooseDesign(session)
     => t | nil
```

Description

Displays the Choosing Design form that lets you select the design to simulate.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevEditSelectedAnas

```
sevEditSelectedAnas(session)
     => t | nil
```

Description

Displays the Choosing Analyses form that lets you select and edit the analyses.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevEditSelectedVars

```
sevEditSelectedVars(session)
     => t | nil
```

Description

Displays the Editing Design Variables form that lets you edit the simulation variables and their values.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevEditSelectedOuts

```
sevEditSelectedOuts(session)
     => t | nil
```

Description

Displays the Setting Outputs form that lets you edit the simulation outputs.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevChangeOutsOnSchematic

```
sevChangeOutsOnSchematic(session setType ?selectionMode)
     => t | nil
```

Description

Sets up for selection of selected output types from the schematic.

Arguments

session	The simulation environment session.
setType	The type of outputs to change - `save`, `march`, or `plot`.
selectionMode	The selection mode for selecting outputs.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevSaveOptions

```
sevSaveOptions(session)
     => t | nil
```

Description

Displays the Save Options form that lets you select what voltages and currents should be automatically saved in the simulation.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevDeleteSelectedAnas

```
sevDeleteSelectedAnas(session)
     => t | nil
```

Description

Deletes the analyses that are currently selected in the Analysis listbox.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevNoAnaSelections

```
sevNoAnaSelections(session)
     => t | nil
```

Description

Indicates whether any analyses are selected in the Analysis listbox.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if no analyses are selected in the analysis listbox.
`nil`	Returns `nil` if there are analyses selected in the analysis listbox.

sevActivateSelectedAnas

```
sevActivateSelectedAnas(session active)
     => t | nil
```

Description

Enables or disables the selected analyses.

Arguments

session	The simulation environment session.
active	Boolean (t/nil) to either activate or deactivate the selected analyses.

Value Returned

t	Returns t if the call is successful.
nil	Returns nil if the call is unsuccessful.

sevDeleteSelectedVars

```
sevDeleteSelectedVars(session)
     => t | nil
```

Description

Deletes any variables that are selected in the Variables listbox.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevNoVarSelections

```
sevNoVarSelections(session)
     => t | nil
```

Description

Indicates whether any variables are selected in the Variables listbox.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if no variables are selected in the Variables listbox.
`nil`	Returns `nil` if variables are selected in the Variables listbox.

sevFindSelectedVars

```
sevFindSelectedVars(session)
     => t | nil
```

Description

Highlights the device on the schematic where the selected variable is used.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevCopyCellViewVariables

```
sevCopyCellViewVariables(session)
     => t | nil
```

Description

Copies the cell view variables and their values into the simulation environment.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevCopyVariablesToCellView

```
sevCopyVariablesToCellView(session)
     => t | nil
```

Description

Copies the simulation session variables and their values into the cellview.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevDeleteSelectedOuts

```
sevDeleteSelectedOuts(session @optional listbox)
     => t | nil
```

Description

Deletes the selected items from the outputs list box.

Arguments

session	The simulation environment session.
listbox	List of items to be deleted from the outputs list box.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevExportOutputsToTxt

```
sevExportOutputsToTxt(session)
     => t | nil
```

Description

Exports the outputs specified in the Output listbox to a text file. This text file can be edited and imported back to ADE.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevImportOutputsFromTxt

```
sevImportOutputsFromTxt(session)
     => t | nil
```

Description

Imports the outputs saved in a text file to the output list box in the main ADE window.

Note: Ensure that you retain the format of the file as it was exported from ADE Output list box using *sevExportOutputsToTxt().*

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevExportOutputsToCSV

```
sevExportOutputsToCSV(sevSession)
     => t | nil
```

Description

Exports the outputs specified in the ADE setup to the CSV file.

Arguments

sevSession	The simulation environment session. For more information, refer to *sevSession*.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

Example

```
sevExportOutputsToCSV(sevSession)
=> t
```

sevExportOutputsToFile

```
sevExportOutputsToFile(sevSession)
     => t | nil
```

Description

Exports the output from the ADE setup to the text or CSV file, which is specified by the .cdsenv variable `outputsImportExportVersion`. If the value of `outputsImportExportVersion` variable is greater than `1.0`, then the function generates the output to the CSV file, else it generates the output to the text file.

Note: You can set the value of the .cdsenv variable `outputsImportExportVersion` using the command: `asimenv.misc outputsImportExportVersion float value`.

Arguments

sevSession	The simulation environment session. For more information, refer to *sevSession*.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

Example

```
envSetVal("asimenv.misc" "outputsImportExportVersion" 'float 1.1)
sevExportOutputsToFile(sevSession)
=> t
```

Generates the CSV file with the outputs specified in the ADE L session `sevSession`.

sevImportOutputsFromCSV

```
sevImportOutputsFromCSV(sevSession)
     => t | nil
```

Description

Imports the outputs from the CSV file to the ADE setup.

Arguments

sevSession	The simulation environment session. For more information, refer to *sevSession*.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

Example

```
sevImportOutputsFromCSV(sevSession)
=> t
```

sevImportOutputsFromFile

```
sevImportOutputsFromFile(sevSession)
     => t | nil
```

Description

Imports the output to the ADE setup from the text or CSV file, which is specified by the .cdsenv variable `outputsImportExportVersion`. If the value of `outputsImportExportVersion` variable is greater than `1.0`, then the function imports the outputs from the CSV file, else it imports the outputs from the text file.

Note: You can set the value of the .cdsenv variable `outputsImportExportVersion` using the command: `asimenv.misc outputsImportExportVersion float value`.

Arguments

sevSession	The simulation environment session. For more information, refer to *sevSession*.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

Example

```
envSetVal("asimenv.misc" "outputsImportExportVersion" 'float 1.1)
sevImportOutputsFromFile(sevSession)
=> t
```

Import outputs from the CSV file in the ADE L session `sevSession`.

sevNoOutSelections

```
sevNoOutSelections(session)
     => t | nil
```

Description

Indicates whether any output is selected in the Outputs listbox.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if no outputs are selected.
`nil`	Returns `nil` if an output is selected.

sevSetPropertyForSelectedOuts

```
sevSetPropertyForSelectedOuts(session property value)
     => t | nil
```

Description

Sets the property to the specified value on the selected outputs.

Arguments

session	The simulation environment session.
property	The property name to set on the selected outputs.
value	The value to set the property to.

Value Returned

t	Returns t if the call is successful.
nil	Returns nil if the call is unsuccessful.

sevSimulator

```
sevSimulator(session)
     => simulator_name
```

Description

Displays the name of the simulator used in the session as a string.

Arguments

session	The simulation environment session.

Value Returned

simulator_name	Returns the name of the simulator in the session.

sevRunEngine

```
sevRunEngine(session)
     => t | nil
```

Description

Runs a simulation from the simulation environment session.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevStopEngine

```
sevStopEngine(session)
     => t | nil
```

Description

Stops the currently running simulation that is tied to the session.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevIsContinuable

```
sevIsContinuable(session)
     => t | nil
```

Description

Indicates whether the currently run simulation can be continued from its stopping point.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevSetEngineOptions

```
sevSetEngineOptions(session type)
     => t | nil
```

Description

Displays the Engine Options form for the selected type for editing.

Arguments

session	The simulation environment session.
type	The type of engine options to edit.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevNetlistFile

```
sevNetlistFile(session type)
     => t | nil
```

Description

Creates the selected type of netlist file.

Arguments

session	The simulation environment session.
type	The type of netlist file to create. For socket netlisting, you may choose between `createRaw`, `displayRaw`, `createFinal`, or `displayFinal`. For direct netlisting, the options are `create`, `display`, and `recreate`.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevOpenEncap

```
sevOpenEncap(session)
     => t | nil
```

Description

Opens the command type in window that lets you enter commands directly to the simulator.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevViewSimulatorOutput

```
sevViewSimulatorOutput(session)
     => t | nil
```

Description

Displays the simulation output log.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevNoOutputLog

```
sevNoOutputLog(session)
     => t | nil
```

Description

Indicates whether the simulation output log exists.

Arguments

session	The simulation environment session.

Value Returned

`t`	Return `t` if no simulation output log exists.
`nil`	Returns `nil` if the simulation output log does exist.

sevConvergence

```
sevConvergence(session type)
     => t | nil
```

Description

Displays the form for setting up the selected type of convergence aid.

Arguments

session	The simulation environment session.
type	The type of convergence aid to set up. The value can be any of these: `'storeRestore`, `'transientStoreRestore`, `'nodeSet`, `'initialCondition`, or `'forceNode`.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevNoResults

```
sevNoResults(session @optional type)
     => t | nil
```

Description

Indicates whether the specified results exist.

Arguments

session	The simulation environment session.
type	The type of results to check for. The type of results to check existence of. The values can be any of these: 'dc, 'dc_op, 'dc_sens, 'dc_op_sens, 'ac, 'ac_sens, 'noise, 'tran, 'tran_op, 'tran_ic, 'tran_sens, 'xf, 'sparam, 'spss.td.pss, 'spss.fd.pss, 'td.pss, 'td.envlp, 'tran.pss, 'fd.pss, 'fd.envlp, 'fd.pdisto, 'fi.pdisto, 'tdr, 'hb, 'four, 'disto, 'model, 'instance, 'output, 'design_variables, 'summary, 'hb_noise, 'osc.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevNoPlottableOutputs

```
sevNoPlottableOutputs(session)
     => t | nil
```

Description

Indicates whether there are any plottable outputs.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if there are no plottable expressions.
`nil`	Returns `nil` if there are plottable expressions.

sevCircuitCond

```
sevCircuitCond(session)
     => t | nil
```

Description

Displays the Circuit Conditions form that lets you set or display any special circuit conditions.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevNoDesign

```
sevNoDesign(session)
     => t/nil
```

Description

Indicates whether a design is tied to the simulation environment.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevSetSimDataDir

```
sevSetSimDataDir( session dir )
     => t/nil
```

Description

Loads the results for the specified simulation environment session and results directory.

Arguments

session	The simulation environment session.
dir	The complete path of the results directory where the data is stored. The path should include the name of the results directory.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

Example

```
sevSetSimDataDir( 'sevSession1 "~/simulation/ampTest/spectre/myResults" )
=> t
```

Loads the results from the results directory, `myResults`, at the path `~/simulation/ampTest/spectre` for the `sevSession1` session.

sevSaveResults

```
sevSaveResults(session)
     => t/nil
```

Description

Displays the *Save Results* form that lets you save the current results.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevSelectResults

```
sevSelectResults(session)
     => t/nil
```

Description

Displays the *Select Results* form that lets you select and load a previously saved set of results.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevDeleteResults

```
sevDeleteResults(session)
     => t | nil
```

Description

Displays the Delete Results form that lets you delete a set of previously saved results.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevEditPlottingOptions

```
sevEditPlottingOptions(session)
     => t | nil
```

Description

Displays the *Setting Plotting Options* form that lets you edit the plotting and printing options.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevPlotAllOutputs

```
sevPlotAllOutputs(session)
     => t | nil
```

Description

Plots all enabled plottable outputs.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevNoPlottableSignals

```
sevNoPlottableSignals(session)
     => t | nil
```

Description

Indicates whether there are any plottable outputs.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevPlotSignals

```
sevPlotSignals(session type ?disableRedraw)
     => t | nil
```

Description

Plots the specified type of signals that exist as outputs.

Arguments

session	The simulation environment session.
type	The type of signals to plot. The value can be any of these: `'tran`, `'ac`, `'dc`, or `'noise`.
disableRedraw	Flag to disable redraw of plots.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevEvaluateAndPlotExpressions

```
sevEvaluateAndPlotExpressions(session)
     => t | nil
```

Description

Evaluates and plots all the output expressions.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevNoPlottableExpressions

```
sevNoPlottableExpressions(session)
     => t | nil
```

Description

Indicates whether there are any plottable expressions.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevPrintResults

```
sevPrintResults(session type)
     => t | nil
```

Description

Displays the Print Results form that lets you print the selected type of simulation results.

Arguments

session	The simulation environment session.
type	The type of results to print. The value can be any of these: 'dcOpPoints, 'tranOpPoints, 'modelParameters, 'sensitivities, 'noiseParameters, 'noiseSummary, 'dcNodeVoltages, or 'tranNodeVoltages.

Value Returned

t	Returns t if the call is successful.
nil	Returns nil if the call is unsuccessful.

sevAnnotateResults

```
sevAnnotateResults(session type)
     => t | nil
```

Description

Annotates the selected results to the schematic window.

Arguments

session	The simulation environment session.
type	The type of results to annotate to the schematic. The value can be any of these: 'componentParameters, 'modelParameters, 'dcOpPoints, 'tranOpPoints, 'netNames, 'dcNodeVoltages, 'tranNodeVoltages, 'defaults, 'dcTermCurrents, 'tranTermCurrents, 'pinNames, or 'voltageLevels.

Value Returned

t	Returns t if the call is successful.
nil	Returns nil if the call is unsuccessful.

sevParametricTool

```
sevParametricTool(session)
     => t | nil
```

Description

Opens the Parametric Analysis tool.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevCornersTool

```
sevCornersTool(session)
     => t | nil
```

Description

Opens the Corners tool.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevMonteCarloTool

```
sevMonteCarloTool(session)
     => t | nil
```

Description

Opens the Analog Statistical Analysis tool.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevOptimizationTool

```
sevOptimizationTool(session)
     => t | nil
```

Description

Opens the Optimization tool.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevOpenCalculator

```
sevOpenCalculator()
     => t | nil
```

Description

Opens the Calculator.

Arguments

None.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevOpenDRLBrowser

```
sevOpenDRLBrowser()
     => t | nil
```

Description

Opens the Browse Project Hierarchy window.

Arguments

None.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevOpenPlotWindow

```
sevOpenPlotWindow(session)
     => t | nil
```

Description

Opens a plot window.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevOpenPrintWindow

```
sevOpenPrintWindow(session)
     => t | nil
```

Description

Opens a print window.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevOpenJobMonitor

```
sevOpenJobMonitor()
     => t | nil
```

Description

Opens the Job monitor.

Arguments

None.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevIcon

```
sevIcon(name)
     => t | nil
```

Description

Displays the icon object corresponding to the specified name.

Arguments

name	The name of the icon for the fixed menu/icon bar. This depends on the created icons. The existing Cadence simulation environment icons are: `'circuit`, `'emitterFollower`, `'ruler`, `'alpha`, `'analyses`, `'acdc`, `'dcTranAc`, `'acTranDc`, `'knobPanel`, `'variables`, `'pin`, `'outputs`, `'pencilEraser`, `'scissors`, `'vcrPlay`, `'trafficGoSmall`, `'trafficGo`, `'trafficYellow`, `'vcrStop`, `'trafficStopSmall`, `'trafficStop`, `'waveform`, `'fx`, or `'sine`.

Value Returned

o_icon	Returns the icon object that can be used in the icon strip.
`nil`	Returns `nil` if the named icon cannot be found.

sevDeleteSelections

```
sevDeleteSelections(session)
     => t | nil
```

Description

Deletes the selected items in any of the simulation environment list boxes.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevWhatsNew

```
sevWhatsNew()
     => t | nil
```

Description

Opens the Whats New window for the simulation environment.

Arguments

None.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevAboutTool

```
sevAboutTool(toolname)
     => t | nil
```

Description

Generates and displays the standard About DFII window for the tool with the tool name in the message.

Arguments

toolname	Name of the tool calling `sevAboutTool`.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevStartSession

```
sevStartSession(?design ?lib ?cell ?view ?schematic)
     => t | nil
```

Description

Starts the Virtuoso Analog Simulation Environment session tied to the specified design. It will try the design first, then lib/cell/view, and finally it will try the schematic. If none of these is specified, it will start a skeleton session that will not be able to do anything until a design has been tied to the session. The lib/cell/view arguments must be specified, otherwise they are ignored.

Arguments

design	The design object. The default value is `nil`.
lib	The library name. The default value is `nil`.
cell	The cell name. The default value is `nil`.
view	The view name. The default value is `nil`.
schematic	The schematic object. The default value is `nil`.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevEditModels

```
sevEditModels(session)
     => t | nil
```

Description

Displays the Model Library Setup form that lets you edit the Spectre direct model libraries.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevSetupStimuli

```
sevSetupStimuli(session)
     => t | nil
```

Description

Displays the Setup Analog Stimuli form that lets you specify the circuit stimuli.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevSetupSimulationFiles

```
sevSetupSimulationFiles(session)
     => t | nil
```

Description

Displays the Simulation Files Setup form that lets you specify the simulation files and paths for Spectre direct simulation.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevNetlistAndRun

```
sevNetlistAndRun(session)
     => t | nil
```

Description

Forces the circuit to netlist and then runs the simulation.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevRun

```
sevRun(session)
     => t | nil
```

Description

Runs a simulation using the current netlist.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevNetlistAndDebug

```
sevNetlistAndDebug(session)
     => t | nil
```

Description

Forces the circuit to netlist and runs the AHDL debugger.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevDebug

```
sevDebug(session)
     => t | nil
```

Description

Runs the AHDL debugger using the current netlist.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevLMGTool

```
sevLMGTool(session)
     => t | nil
```

Description

Opens the Transmission Line Modeler tool.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevPKGTool

```
sevPKGTool(session)
     => t | nil
```

Description

Opens the RFIC Package Modeler tool.

Arguments

session	The simulation environment session.

Value Returned

t	Returns t if the call is successful.
nil	Returns nil if the call is unsuccessful.

sevKmodelTool

```
sevKmodelTool(session)
     => t | nil
```

Description

Opens the RIFC Modeler for Cierto SPW tool.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevPCMTool

```
sevPCMTool(session)
     => t | nil
```

Description

Opens the Spiral Inductor Modeler tool.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevBPMTool

```
sevBPMTool(session)
     => t | nil
```

Description

Opens the Bond Pad Modeler tool.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevBALMTool

```
sevBALMTool(session)
     => t | nil
```

Description

Opens the Transformer Modeler tool.

Arguments

session	The simulation environment session.

Value Returned

t	Returns t if the call is successful.
nil	Returns nil if the call is unsuccessful.

sevActiveSelectedAna

```
sevActiveSelectedAna(o_session)
     => l_analyses | nil
```

Description

Returns the list of selected analyses that are currently enabled.

Arguments

o_session	The simulation environment session.

Value Returned

l_analyses	List of the selected, enabled analyses.
nil	Returns nil if there are no analyses selected in the analysis listbox or all the selected analyses are not enabled.

sevNonActiveSelectedAna

```
sevNonActiveSelectedAna(o_session)
     => l_analyses | nil
```

Description

Returns the list of selected analyses that are currently not enabled.

Arguments

o_session	The simulation environment session.

Value Returned

l_analyses	The list of selected, non-enabled analyses.
nil	Returns nil if there are no analyses selected in the analysis listbox or all the selected analyses are enabled.

sevSession

```
sevSession(
     o_entity
     )
     => t_session | nil
```

Description

Displays the session ID of the simulation environment such as the ADE L window, the schematic window associated with ADE L, or a form launched from ADE L.

Arguments

o_entity	The object with which the simulation environment session is associated. For example, form or window.

Value Returned

t_session	Returns the session ID of the current simulation environment window.
nil	Returns `nil` if there is no simulation environment session object currently tied to the entity.

Examples

Example 1

```
sevSession(hiGetCurrentWindow())
=> sevSession2
```

When you specify `hiGetCurrentWindow()` as the object, the function returns the session ID of the current ADE L window.

Example 2

```
sevSession(hiGetCurrentForm())
=> sevSession2
```

When you specify `hiGetCurrentForm()` as the object, the function returns the session ID of the ADE L window associated with the currently opened form.

Example 3

```
sevSession(window(8))
=> sevSession2
```

When you specify the `window(<`*`windowID`*`>)` as the object, the function returns the session ID associated with the *windowID*.

sevSetTopSaveDir

```
sevSetTopSaveDir(o_session)
     => t
```

Description

This function sets the ADE L Save State directory for the simulation environment session.

Arguments

o_session	The simulation environment session.
t_dir	The directory path to be set as the ADE L Save State directory.

Value Returned

t	Returns t if successful.

sevTopSaveDir

```
sevTopSaveDir(o_session)
     => t_dir
```

Description

Displays the current ADE L Save State directory path.

Arguments

o_session	The simulation environment session.

Value Returned

t_dir	Returns the ADE L Save State directory path.

sevDisplayViolations

```
sevDisplayViolations(s_sevSession)
     => o_form
```

Description

Displays the violations form.

Arguments

s_sevSession	The simulation environment session.

Value Returned

o_form	Displays the violations form.

Example

```
sevDisplayViolations(session)
```

sevNoViolationsFound

```
sevNoViolationsFound(s_sevSession)
     => t | nil
```

Description

Determines if any violation file has been found in the results.

Arguments

s_sevSession	The simulation environment session.

Value Returned

t	Returns t if no violation files are found in the results.
nil	Returns nil if violations files are found in the results.

Example

```
sevNoViolationsFound(s_sevSession)
=> t
```

sevParasiticsDisplayed

```
sevParasiticsDisplayed(session)
     => t | nil
```

Description

Determines whether the *Show Parasitics* menu will be disabled when the DC Operating Point results are available.

Arguments

session	The simulation environment session.

Value Returned

t	Returns t if the *Show Parasitics* menu is disabled.
nil	Returns nil if the *Show Parasitics* menu is not disabled.

Example

```
sevParasiticsDisplayed(session)
```

sevParasiticsNotDisplayed

```
sevParasiticsNotDisplayed(session)
     => t | nil
```

Description

Determines if the *Hide Parasitics* menu will be disabled when the DC Operating Point results are available.

Arguments

session	The simulation environment session.

Value Returned

t	Returns t if the *Hide Parasitics* menu is disabled.
nil	Returns nil if the *Hide Parasitics* menu is not disabled.

Example

```
sevParasiticsNotDisplayed(session)
```

sevDevChecking

```
sevDevChecking(session)
     => o_form | nil
```

Description

Displays the Analog Design Environment *Device Checking Setup* form.

Arguments

s_sevSession	The simulation environment session.

Value Returned

o_form	Returns the form object and displays the *Device Checking Setup* form if the call is successful.
nil	Returns nil if the call is unsuccessful.

Example

```
sevDevChecking(sevSession)
=> formStruct@0x38930e8
```

sevSetSolver

```
sevSetSolver(session)
     => t | nil
```

Description

Displays the *Choose Solver* form, which lets you select a solver.

Arguments

session	The simulation environment session.

Value Returned

t	Returns t and displays the solver form if the call is successful.
nil	Returns nil if the call is unsuccessful.

sevSetConnectModules

```
sevSetConnectModules(session)
     => t | nil
```

Description

Displays the *Connect Rules* form that allows you to select built-in or user-defined connect rules.

Arguments

session	The simulation environment session.

Value Returned

t	Returns t if the call is successful.
nil	Returns nil if the call is unsuccessful.

sevInvokeNCBrowse

```
sevInvokeNCBrowse(session)
     => t | nil
```

Description

Displays the *NCBrowse* window.

Arguments

session	The simulation environment session.

Value Returned

t	Returns t if the call is successful.
nil	Returns nil if the call is unsuccessful.

sevInvokeSimvision

```
sevInvokeSimvision(session)
     => t | nil
```

Description

Displays the *SimVision Waveform* window.

Arguments

session	The simulation environment session.

Value Returned

t	Returns t if the call is successful.
nil	Returns nil if the call is unsuccessful.

sevInvokeSimvisionDebugger

```
sevInvokeSimvisionDebugger(session)
     => t | nil
```

Description

Displays the *SimVision Debugger* interface with GUI options during an AMS session.

Arguments

session	The simulation environment session.

Value Returned

t	Returns t if the call is successful.
nil	Returns nil if the call is unsuccessful.

sevNoLog

```
sevNoLog(session type)
     => t | nil
```

Description

Checks if the specified log file exists for the AMS interface.

Arguments

session	The simulation environment session.
type	The type of log file - compiler, elaborator, or simulator log files. Valid Values: ‘compiler’, ‘elaborator’, and ‘simulator’.

Value Returned

t	Returns t if the specified log file is not found.
nil	Returns nil if the specified log file is found.

sevViewNetlisterLog

```
sevViewNetlisterLog(session)
     => t | nil
```

Description

Displays the AMS netlister log file.

Arguments

session	The simulation environment session.

Value Returned

t	Returns t if the call is successful.
nil	Returns nil if the call is unsuccessful.

sevViewCompilerLog

```
sevViewCompilerLog(session)
     => t | nil
```

Description

Displays the AMS simulation compiler log file. The log file can be either `ncvlog.log` or `ncvhdl.log` depending on the contents of the AMS design.

Arguments

session	The simulation environment session.

Value Returned

`t`	Returns `t` if the call is successful.
`nil`	Returns `nil` if the call is unsuccessful.

sevViewElabLog

```
sevViewElabLog(session)
     => t | nil
```

Description

Displays the AMS simulation elaborator log file.

Arguments

session	The simulation environment session.

Value Returned

t	Returns t if the call is successful.
nil	Returns nil if the call is unsuccessful.

sevViewNcverilogLog

```
sevViewNcverilogLog(session)
     => t/nil
```

Description

Displays the AMS simulation NcVerilog log file.

Arguments

session	The simulation environment session.

Value Returned

t	Returns t if the call is successful.
nil	Returns nil if the call is unsuccessful.

sevViewSimLog

```
sevViewSimLog(session)
     => t | nil
```

Description

Displays the AMS simulator log file.

Arguments

session	The simulation environment session.

Value Returned

t	Returns t if the call is successful.
nil	Returns nil if the call is unsuccessful.

sevReturnVariablesWithEmptyValues

```
sevReturnVariablesWithEmptyValues(o_session)
     => t_string | nil
```

Description

Returns the set of variables in a session with empty values.

Arguments

o_session	The simulation environment session.

Value Returned

t_string	Returns a string containing variable names separated by comma. For example, “Var1, Var2”.

sevAddExpression

```
sevAddExpression(o_session t_expressionname t_expression)
     => t | nil
```

Description

This function takes the name and expression as a string and adds a corresponding output in ADE session.

Arguments

session	A valid session passed as a string or symbol.
t_expressionname	Name of the expression to be added
t_expression	Expression passed as a string.

Value Returned

`list(nil errorString)`	Returns `list(nil errorString)` if expression adding failed.
`list(t outputStruct)`	Returns `list(t outputStruct)` if expression added successfully.

Example

```
sevAddExpression("sevSession1" "a" "1+1")
```

sevGetExpressions

```
sevGetExpressions(o_session @key (t_axlTestName nil) (t_namedOnly nil)
     => l_list
```

Description

Returns all the expressions in the specified ADE/ADEXL session.

Arguments

o_session	The simulation environment session.
t_axlTestName	Name of the test. If passed, method returns expressions to the corresponding test.
t_expression	If passed, returns only named expressions.

Value Returned

l_list	List of name value pairs.

Example

```
sevGetExpressions("sevSession1" ?axlTestName "aaa" ?namedOnly t)
```

Returns all the named expressions corresponding to the test `aaa` in `sevSession1`.

sevDeleteSelectedSubckts

```
sevDeleteSelectedSubckts(
     t_sevSession [l_instanceList]
     )
     => t | nil
```

Description

Deletes the subcircuit instances selected in the *Save By Subckt Instances* pane of the simulation window.

Arguments

t_sevSession	The simulation environment session. For more information, see *sevSession*.
l_instanceList	List of subcircuit instances to be deleted from the *Save By Subckts Instances* pane.

Value Returned

`t`	Returns `t` if the selected subcircuit instances are deleted.
`nil`	Returns `nil` if the specified simulation environment session object is invalid, or no instance is selected in the *Save By Subckt Instances* pane of the simulation window.

Examples

The following example deletes all the selected subcircuit instances in the simulation environment session, `sevSession1`:

```
sevSession(hiGetCurrentWindow())
=> sevSession1

sevDeleteSelectedSubckts('sevSession1)
=> t
```

21

CDF Functions

The Component Description Format (CDF) describes the parameters and the attributes of parameters of individual components and libraries of components. The Virtuoso CDF tools let you create, edit, and delete CDF data of components. For details on using the graphical user interface of these tools, see the *Component Description Format User Guide*.

Using the CDF SKILL functions, you can create routines to perform the operation on many components at once, automating your handling of CDF descriptions. This chapter describes the functions that let you perform these operations.

CDF SKILL Function Elements

CDF SKILL functions use or operate on data IDs, parameters, parameter attributes, expressions, and global variables. This section describes each of these elements in the context of SKILL.

Cell and Library Data IDs

Before working on a CDF description, you must specify the data ID of the cell or library. You can get the data ID for the cell or library you are using with the `ddGetObj()` function. With SKILL versions earlier than 4.4, you used the `dmFindLib()` and `dmFindCell()` functions. You must assign the value returned by these functions to a variable that you create. For example, to operate on the *analogLib* library, you must first create the variable *mylib* with this assignment:

```
mylib = ddGetObj("analogLib")
```

When creating the variable for the object ID of a cell, you must specify both the library name and the cell name:

```
test_cell = ddGetObj("analogLib" "schottky")
```

or

```
test_cell = ddGetObj(mylib "schottky")
```

if you have already defined *mylib*.

You must then use *mylib* when a SKILL function requires a *d_id* or *d_libId*, and test_cell when a SKILL function requires a *d_id* or *d_cellId*.

You can use this data ID to access information about the description by using the right arrow (-> or ~>) operator. For example

```
test_cell -> name
```

returns the ID

```
"schottky"
```

Data Objects

CDF descriptions are represented by *cdfDataId* objects, which are SKILL objects that you can manipulate like other SKILL database objects. Just like data IDs, you assign the *cdfDataId* of a particular CDF description to a variable that you create. The most common variable name is *cdfDataId*. To create a new CDF data object for a new CDF description, use this type of ID assignment:

```
newCellId = cdfCreateUserCellCDF(test_cell)
cdf:25092160
newCellId->type
"userCellData"
```

To access an existing CDF data object for an existing CDF description, use this type of ID assignment:

```
baseCell = cdfGetBaseCellCDF(test_cell)
baseCell->dataFile->value
"bjt"
```

newCellId and *baseCell* are arbitrary names that you create.

Although you can add any information to a *cdfDataId*, the object has specific fields to hold the information that it maintains. These fields are described in the following section.

id

The database object (cell ID or library ID) to which the *cdfDataId* is attached. This field is not editable.

type

There are seven types of *cdfDataId*:

baseLibCDF	base-level library CDF
userLibCDF	user-level library CDF
effLibCDF	effective-level library CDF
baseCellCDF	base-level cell CDF
userCellCDF	user-level cell CDF
effCellCDF	effective-level cell CDF
effInstCDF	effective-level instance CDF

This field is not editable.

parameters

List of parameters attached to this *cdfDataId*. This field is not editable.

doneProc

An optional procedure name (a string) that is evaluated after any change to a parameter on a component instance. You can use *doneProc* for post-processing. The procedure must take a single argument: the instance that has been modified.

The *doneProc* callback function can be used to modify the `cdfgForm` fields that are editable.

The default value is `nil`. This field is editable.

formInitProc

An optional procedure name. If specified, the procedure is executed when the contents of the CDF are displayed on a form. The default value is *nil*. This field is editable.

This procedure runs when you use the *Add Instance* and *Edit Properties* commands. You can use the procedure for preprocessing CDF data. This procedure must take a single argument, the *cdfDataId* being added to the form.

Note: When you modify a parameter field value using the *formInitProc*, the Edit Properties Object command for the schematic application is not aware of the modification and does not update the changes made with *formInitProc*. You can avoid this problem by setting the variable *cdfgForm->cdfModified* to the value *t*.

Displaying Parameters

The following fields control the display of parameters on a form.

fieldWidth	Width of a field	Default = 350 pixels
fieldHeight	Height of a field	Default = 35 pixels
buttonFieldWidth	Width of a button field	Default = 340 pixels
promptWidth	Width of a prompt	Default = 175 pixels

Parameters

CDF parameters are represented by *cdfParamId* objects, which are SKILL objects that you can manipulate like other SKILL database objects. You can use a *cdfParamId* to access

information about a parameter by using the right arrow (->) operator. Although you can add any information to a *cdfParamId*, the object has fields to hold information that it maintains.

In addition to getting the *cdfParamId*s through the parameters field of a *cdfDataId*, you can also access a parameter by specifying:

```
cdfDataId->paramName
```

paramName is the name of the parameter. However, you must not create any user-defined properties on a *cdfDataId* that conflict with the name of a parameter on the *cdfDataId*.

use

Specifies whether to use this parameter. The *use* attribute is context-specific and is evaluated when necessary. In this field, you can specify parameters that you are not using because of the value of other parameters, or because of the function you are using. The *use* field must be a string.

- If the field evaluates to *non-nil*, the system uses the parameter.
- If a value for the field exists, the system ignores the value.
- If the field evaluates to *nil*, the system does not use the parameter.
- If you do not specify this field, *t* is assumed, so the system always uses the field.

The “Global Variables” section describes several global variables for constructing the *use* expression.

This field is editable.

paramType

Type of the parameter. Valid values for this field are

```
"string"
"int"
"float"
"radio"
"cyclic"
"boolean"
"button"
"netSet"
```

The quotation marks are required. CDF checks that the parameter value matches the type specified. You must specify this field.

This field is not editable.

Note: Use the netSet type to store inherited connections in CDF. Inherited connections allow you to selectively override global signals in designs created in Virtuoso Schematic Editor. The override information is communicated through net expressions and netSet properties. For more information on netSet properties refer to *Inherited Connections Flow Guide* and *Virtuoso Schematic Editor L User Guide*.

defValue

Default value of the parameter. You must specify this field.

This field is editable.

value

Current value of the parameter.

This field is editable.

prompt

Prompt that appears on a form field when an application asks for the value of this parameter. The value for this field must be a string. You must specify this field.

This field is editable.

choices

Possible values (a list of strings) of a *cyclic* or *radio* type parameter. Do not use this field for other types of parameters.

This field is editable.

units

Unit type of the parameter. You often modify parameters that represent resistance, capacitance, length, time, power, and so forth, and expect to see the parameter value scaled

to appropriate units (for example, *pF*, *uM*, *dBm*). Valid values for this field include the following strings:

```
"resistance"
"capacitance"
"inductance"
"conductance"
"time"
"frequency"
"power"
"powerDB"
"lengthMetric"
"lengthEnglish"
"angle"
"voltage"
"current"
"temperature"
```

The quotation marks are required. This field determines the unit's suffix and scale factor to use when displaying the parameter value. For example, by setting a parameter's units field to *capacitance,* the parameter value is displayed as *5 pF* instead of *5e-12*.

editable

Specifies whether a parameter is for display only. If set to *nil*, the field specifies a displayed parameter that you cannot change. (Other parameters' callbacks can change the parameter.) Use this field only on *int*, *float*, and *string* type fields.

The *editable* field must be a string. This field is evaluated when necessary.

- If the field evaluates to *non-nil*, the parameter is editable.
- If the field evaluates to *nil*, the parameter is grayed out when displayed as a form field.
- If you do not specify the field, *t* is assumed, meaning that the field is editable.

callback

SKILL code to be executed whenever the value of a parameter changes. Using *callback*, you can cause any parameter's value to affect any other parameter's value. The value of this field must be a string.

This field is optional. If you do not use *callback*, no callback is executed.

The "Global Variables" section describes several global variables you can use in the *callback* field.

parseAsNumber

String type parameter whose value can evaluate to a floating-point number. These types of parameters often occur for circuit level simulators that allow component parameters to be either numbers or expressions containing variables.

If you specify this field, the system parses the value of the parameter to check if the value is a number.

- If the value is a number, the system converts the string to a floating point number, converts the string to the most efficient notation (taking into account any units field specified), then reconverts the number into a string.
- If the parameter value is not a number, the system does no conversion.

For example, if the parameter value is the string *5.4e-12* and the parameter value has a units field of *capacitance* (that is, the suffix is F), the parameter value is converted to the string *5.4p* before being displayed. If, however, the parameter value is *c1*, no conversion takes place.

dontSave

Specifies that the parameter cannot be stored in any instance that corresponds to the component.

The *dontSave* field must be a string. The system evaluates the field as necessary. If the field evaluates to non-*nil*, the parameter is not stored. If you do not specify the field, *nil* is assumed and the parameter value is stored.

parseAsCEL

If set to *yes*, the associated CDF parameter is processed as a CDF Expression Language (CEL) expression. The parameter must be a string. If the expression resolves to a numeric value (the usual case), the *parseAsNumber* flag should also be set to *yes*.

storeDefault

Specifies whether the parameter default value is saved as a property of the instance. All tools based on the Cadence Analog Design Environment software use the CDF to find default values if no property exists on an instance.

If set to *no* (the default) or *don't use*, a property is not saved on the instance when the parameter value is the default. Also, if the default value of the parameter changes, all instances that use the default automatically get the new default value. (To see the change in an open window, you must select *Window – Redraw* from the Cadence menu.)

If set to *yes* and the parameter value is set to the same value as the default, a property is saved on the instance. One disadvantage of this attribute is that if the default value of a parameter changes, the instances that use the default do not automatically change to the new default. If you are a user of the Open Simulation Software (OSS) system, and you used to set this attribute to *yes* because you could only netlist using instance properties, you can now set it to *no* because OSS users now have the option of using CDF.

display

Determines if this parameter is displayed in forms that display CDF parameters, such as the Edit Object Properties form or the Add Instance form. You must enter `t`, `nil`, or a SKILL expression that evaluates to `t` or `nil` in this field to determine if this parameter is to be displayed. If the field evaluates to non-`nil` (the default), the parameter is displayed. If the field evaluates to `nil`, the parameter is not displayed.

Example

You can use the *cdfDataId baseCell* from the previous example to examine the parameters in the base-level cell CDF description of the Schottky transistor in the analog library. For example, that cell has a parameter *m*. You can access information about *m* in the following manner:

```
baseCell->m->prompt
"Multiplier"
baseCell->m->editable
```

```
nil
baseCell->m->display
"artParameterInToolDisplay('m)"
baseCell->m->paramType
"string"
```

Expressions

CEL (CDF Expression Language) is another name for the Analog Expression Language (AEL) that works with CDF parameters. For information on AEL, refer to the *Analog Expression Language Reference*. With AEL you can express a value as a mathematical expression instead of a single, fixed value. In such an expression, symbolic names such as *sheetResistivity* can refer to values that are computed and set on one of the following:

- CDF parameter on the same design component
- CDF parameter on the parent design component

Global Variables

CDF parameter values interact with each other, and one parameter's value can affect the existence of another. This feature is implemented primarily through the *use* and *callback* fields of the parameters section. The following global variables, which you can access, are set whenever parameter fields are evaluated.

cdfgData

CDF data for the component in use. You can use this variable in the `doneProc` or `formInitProc` callback functions to either read or write data for a component. Use the *`value`* field to get the current value of any parameter in the CDF description.

- For creating an instance, set the field to the last value you used when you created a component of this type.
- For editing, set the field to the value for the component you are editing.

For example, you might set the *use* field for the *resistance* parameter to

```
"cdfgData->resType->value == \"ideal\""
```

implying that the *resistance* parameter should be used only if the resistor type is set to *ideal*. (== is an equality test, not an assignment.)

When setting the value of a parameter, set

```
cdfgData->paramName->value = paramValue
```

If you use this setting, in the future you should be able to use the property list editor to edit the parameter values directly without going through the form.

cdfgForm

Form on which the CDF data is displayed, if there is one. You can modify data stored in *cdfgData* in the CDF *formInitProc*, or by modifying the *cdfgForm* fields with a callback function. When doing this, set the Boolean variable *cdfgForm -> cdfModified* to *t*.

gLabelsNumNotation

Displays the cdsTerm and cdsParam values in different notations, such as scientific or engineering. The syntax of *gLabelsNumNotation* is as follows:

```
gLabelsNumNotation = 'suffix
```

The possible values can be `'suffix`, `'scientific`, `'engineering`, `'simple` or `'default`.

The `default` value displays the labels according to the existing setting. For example, the nmos symbol shows the engineering notation, `300e-3` for `w`, by default.

To set the number of significant digits, use the `aelPushSignifDigits` function as follows:

```
aelPushSignifDigits(10)
```

Creating Descriptions

This section describes the functions that operate on CDF descriptions. You can create base and user CDF descriptions for libraries and cells using the following functions.

cdfCreateBaseLibCDF

```
cdfCreateBaseLibCDF(
     libId
     [?doneProc t_doneProc]
     [?formInitProc t_formInitProc]
     [?fieldWidth x_fieldWidth]
     [?fieldHeight x_fieldHeight]
     [?buttonFieldWidth x_buttonFieldWidth]
     [?promptWidth x_promptWidth]
     )
     => cdfDataId / nil
```

Description

Creates the Base Library CDF that is applied to all the devices in the library. The CDF description is created with no parameters or *simModels*.

Note the following:

- You must open the library in write mode.
- Before using this function, ensure that the base-level CDF description does not already exist for the library. If the CDF description already exists, this function will not update the existing CDF description.

Arguments

`libId`	This is the library ID.
`t_doneProc`	Lets you specify an optional SKILL routine that executes after you change any parameter on the instantiation form.
`t_formInitProc`	Lets you specify an optional SKILL language routine that executes automatically when the component is placed on an instantiation form.
`x_fieldWidth`	Lets you specify the width of a field on the instantiation form. The default width is 350 pixels.
`x_fieldHeight`	Lets you specify the height of a field on the instantiation form. The default height is 35 pixels.
`x_buttonFieldWidth`	Lets you specify the width of a button on the instantiation form. The default width is 350 pixels.
`x_promptWidth`	Lets you specify the width of the prompt on the instantiation form. The default width is 175 pixels.

cdfCreateUserLibCDF

```
cdfCreateUserLibCDF(
     libId
     [?doneProc t_doneProc]
     [?formInitProc t_formInitProc]
     [?fieldWidth x_fieldWidth]
     [?fieldHeight x_fieldHeight]
     [?buttonFieldWidth x_buttonFieldWidth]
```

```
[?promptWidth x_promptWidth]
)
=> cdfDataId / nil
```

Description

Creates the user-level library CDF that is applied to all the devices in the library. The user-level CDF can override entries in the base-level CDF. Therefore, a combination of the base-level CDF and the user-level CDF becomes the effective CDF.

The CDF description is created with no parameters or simulation models.

Note: Before using this function, ensure that the user-level CDF description does not already exist for the library. If the CDF description already exists, this function will not update the existing CDF description.

Arguments

`libId`	This is the library ID.
`t_doneProc`	Lets you specify an optional SKILL routine that executes after you change any parameter on the instantiation form.
`t_formInitProc`	Lets you specify an optional SKILL language routine that executes automatically when the component is placed on an instantiation form.
`x_fieldWidth`	Lets you specify the width of a field on the instantiation form. The default width is 350 pixels.
`x_fieldHeight`	Lets you specify the height of a field on the instantiation form. The default height is 35 pixels.
`x_buttonFieldWidth`	Lets you specify the width of a button on the instantiation form. The default width is 350 pixels.
`x_promptWidth`	Lets you specify the width of the prompt on the instantiation form. The default width is 175 pixels.

cdfCreateBaseCellCDF

```
cdfCreateBaseCellCDF(
     cellId
     [?doneProc t_doneProc]
     [?formInitProc t_formInitProc]
```

```
[?fieldWidth x_fieldWidth]
[?fieldHeight x_fieldHeight]
[?buttonFieldWidth x_buttonFieldWidth]
[?promptWidth x_promptWidth]
)
=> cdfDataId / nil
```

Description

Creates a base-level CDF description for a cell. The CDF description is created with no parameters or simulation models.

Note the following:

- You must open the cell in write mode.
- Before using this function, ensure that the base-level CDF description does not already exist for the cell. If the CDF description already exists, this function will not update the existing CDF description.

Arguments

`cellId`	This is the cell ID.
`t_doneProc`	Lets you specify an optional SKILL routine that executes after you change any parameter on the instantiation form.
`t_formInitProc`	Lets you specify an optional SKILL language routine that executes automatically when the component is placed on an instantiation form.
`x_fieldWidth`	Lets you specify the width of a field on the instantiation form. The default width is 350 pixels.
`x_fieldHeight`	Lets you specify the height of a field on the instantiation form. The default height is 35 pixels.
`x_buttonFieldWidth`	Lets you specify the width of a button on the instantiation form. The default width is 350 pixels.
`x_promptWidth`	Lets you specify the width of the prompt on the instantiation form. The default width is 175 pixels.

cdfCreateUserCellCDF

```
cdfCreateUserCellCDF(
     cellId
     [?doneProc t_doneProc]
     [?formInitProc t_formInitProc]
     [?fieldWidth x_fieldWidth]
     [?fieldHeight x_fieldHeight]
     [?buttonFieldWidth x_buttonFieldWidth]
     [?promptWidth x_promptWidth]
     )
     => cdfDataId / nil
```

Description

Creates a user-level CDF description for a cell. The CDF description is created with no parameters or simulation models.

Note: Before using this function, ensure that the user-level CDF description does not already exist for the cell. If the CDF description already exists, this function will not update the existing CDF description.

Arguments

`cellId`	This is the cell ID.
`t_doneProc`	Lets you specify an optional SKILL routine that executes after you change any parameter on the instantiation form.
`t_formInitProc`	Lets you specify an optional SKILL language routine that executes automatically when the component is placed on an instantiation form.
`x_fieldWidth`	Lets you specify the width of a field on the instantiation form. The default width is 350 pixels.
`x_fieldHeight`	Lets you specify the height of a field on the instantiation form. The default height is 35 pixels.
`x_buttonFieldWidth`	Lets you specify the width of a button on the instantiation form. The default width is 350 pixels.
`x_promptWidth`	Lets you specify the width of the prompt on the instantiation form. The default width is 175 pixels.

cdfCreateParam

```
cdfCreateParam(g_cdfDataId ?name t_name ?type t_type [?defValue g_defValue]
     [?units t_units]
     [?parseAsNumber t_parseAsNumber]
     [?choices l_choices]
     [?prompt t_prompt]
     [?use t_use]
     [?display t_display]
     [?editable t_editable]
     [?dontSave t_dontSave]
     [?callback t_callback]
     [?storeDefault t_storeDefault])
     [?parseAsCEL t_parseAsCEL]
     [?description t_description]

     )
     => cdfDataId | nil
```

Description

Creates a parameter on the specified `cdfDataId` with the specified attributes. The only attributes that are *always* required are the parameter's name and type. If this parameter description is not overriding an existing base-level parameter definition, you must also specify the default value.

Note the following:

- You cannot override a parameter's type. Also, CDF checks that the effective parameter is consistent and valid despite any overrides.
- If the name of a parameter already exists in the CDF description, the CDF description will not be updated.

Note: For more information, see *Component Parameters* in the *Component Description Format User Guide*.

Arguments

g_cdfDataId	The database object that represents the CDF description for the component.
t_name	The name of the parameter.
t_type	The data type of the parameter.

g_defValue	The default value of the parameter.
t_units	The unit suffix and scale factor to use when displaying the parameter value.
t_parseAsNumber	Specifies whether the parameter can be evaluated to a floating-point number. Use this attribute only for string type parameters that contain numeric data. If *t_parseAsNumber* is `yes`, string is converted to a floating-point number, which is then converted the most efficient notation for execution, and then reconverts the floating-point number into a string. If *t_parseAsNumber* is `no` (the default value) or `don't use`, the system does no conversion. Use this setting if the parameter to be defined is a literal that contains numeric characters, such as the name of a file or a model. *t_parseAsNumber* must be used when the value of *t_parseAsCEL* is `yes`.
l_choices	The space- or comma-separated list of selections for a *cyclic* or *radio* data type. This attribute does not apply to other types of parameters. The choices depend on the following two cases: ■ If each choice is a single word, you can separate the choices with spaces. ■ If any choice is a group of words, you must separate each choice with a comma (,). (When using commas, do not leave extra spaces between choices because these spaces become part of the choice value.) A typical entry in the form field can be `choice 1,choice 2,choice 3` Notice that there is no space between the number 1 and the comma, or between the comma and the *c* in *choice*.
t_prompt	The name for the CDF parameter that is displayed on the *Add Instance* or the *Edit Object Properties* form.

t_use	Determines if the parameter is to be used. You can enter Cadence SKILL language expression that evaluates to `t` or `nil` in this field to determine if this parameter is applicable. When the field evaluates to nil, the system never displays the parameter. When it evaluates to non-nil (the default), then based on the value of *t_display*, the system displays the parameter.
t_display	Determines if this parameter is displayed in forms that display CDF parameters, such as the *Edit Object Properties* form or the *Add Instance* form. You must enter `t`, `nil`, or a SKILL expression that evaluates to `t` or `nil` in this field to determine if this parameter is to be displayed. If the field evaluates to non-nil (the default), the parameter is displayed. If the field evaluates to `nil`, the parameter is not displayed.
t_editable	Determines if this parameter can be edited in forms that display CDF parameters, such as the *Edit Object Properties* form or the *Add Instance* form. You can enter a SKILL expression that evaluates to `t` or `nil` in this field to determine if the parameter is editable. If the field evaluates to non-nil (the default), the parameter is editable. If the field evaluates to `nil`, the parameter is not editable. (If not editable, the parameter is grayed so that you can see the value but you cannot edit it.) This field is valid only for `string`, `int`, and `float` data types.
t_dontSave	Determines if the parameter value is to be saved on the instance. This attribute is for programming use only. Typically, you should set this field to `nil` and not use it. If the field evaluates to `nil` (the default), the parameter value is saved as a property on the instance. If the field evaluates to non-nil, the parameter value is not saved as a property on the instance.

Caution

The *t_dontSave* ***attribute overrides the t_storeDefault setting. If*** *t_storeDefault* ***is*** `yes` ***and*** *t_dontSave* ***is*** `yes`***, the default property is not saved.***

t_callback

Specifies a SKILL routine to be executed whenever the value of the parameter changes. The value of this optional field must be a string. The entered value can be an entire function to be executed or a call to a function that is defined elsewhere. If you do not enter anything in this field, the system assumes that there is no callback.

The CDF parameter callback is primarily a GUI-based callback. GUI-based callbacks occur when you modify the values in the parameter form fields. A GUI-based callback is active when the CDF parameters are displayed in the *Add Instance* form or the *Edit Object Properties* form if you use the *Create Instance* or *Edit Properties* commands.

For more information on callbacks, see *Writing Callbacks* in the *Component Description Format User Guide*.

t_storeDefault

Specifies whether to store the default value of a parameter as a parameter attribute on the instance. All tools based on the *Cadence Virtuoso Analog Design* software use the CDF description to find default values if there is no property on an instance.

If set to `no` or `don't use`, the default value that you set for a parameter is not preserved. In this case, if you modify the default value of a parameter, the change is reflected in all the existing instances and new instances of the component.

When the default value of the CDF parameter changes, all the instances including the ones already instantiated are updated with the new default value of the CDF parameter. To see the change in an open window, you must choose *View – Redraw*.

If set to `yes`, a parameter attribute that stores the default value of the parameter is added on the instance. Later if the default value for the parameter is modified, the instances that are already instantiated retain the old default value of the parameter. Only new instantiations have the new default value of the parameter.

One disadvantage of setting this attribute to `yes` is that if the default value of a parameter changes, the existing instances that use the default value do not automatically change to the new default value. That is, they retain the old default value.

Default value: `no`

t_parseAsCEL	Specifies whether the parameter is processed as a CDF Expression Language (CEL) expression. Use this attribute only for string type parameters. If *t_parseAsCEL* is `yes`, the parameter is processed as a CEL expression. (Expressions such as *iPar(x)* or *pPar(x)*, indicating inheritance from an instance parameter or a parent parameter, are processed as CEL expressions.) If *t_parseAsCEL* is `no` (the default), the parameter is not processed as a CEL expression.
t_description	Allows you to specify a tooltip or description for each parameter. The text specified as description will be displayed on the *Edit Object Properties* and, *Create Instance* form, and the *Property Editor* assistant.

Value Returned

`g_cdfDataId`	Returns the database object that represents the CDF description for the component.
`nil`	Returns `nil` if the parameter with the same name already exists.

Example

```
cdfParamId = cdfCreateParam(cdfDataId
?name "resistorType"
?type "radio"
?prompt "Type:"
?defValue "ideal"
?choices list("ideal" "thin film")
?callback "myModelTypeCB()")
```

For more examples of the `cdfCreateParam` function, see *NBSIM Transistor CDF SKILL Description* in the *Component Description Format User Guide*.

Query Descriptions

You can query existing CDF descriptions using the following functions. To use the returned value, you must assign it to a variable that you create.

cdfGetBaseLibCDF

```
cdfGetBaseLibCDF(d_libId
     )
     => cdfDataId / nil
```

Returns the base-level CDF description attached to a library. If one is not defined, it returns `nil`.

cdfGetUserLibCDF

```
cdfGetUserLibCDF(d_libId
     )
     => cdfDataId / nil
```

Returns the user-level CDF description attached to a library. If one is not defined, it returns `nil`.

cdfGetLibCDF

```
cdfGetLibCDF(d_libId
     )
     => cdfDataId / nil
```

Returns the effective CDF description attached to a library. If neither a base- nor user-level CDF description is defined, it returns `nil`. The resulting CDF description represents the overlay of the user-level CDF on the base-level CDF.

cdfGetBaseCellCDF

```
cdfGetBaseCellCDF(d_cellId
     )
     => cdfDataId / nil
```

Returns the base-level CDF description attached to a cell. If one is not defined, it returns `nil`.

cdfGetUserCellCDF

```
cdfGetUserCellCDF(d_cellId
     )
     => cdfDataId / nil
```

Returns the user-level CDF description attached to a cell. If one is not defined, it returns `nil`.

cdfGetCellCDF

```
cdfGetCellCDF(d_cellId
     )
     => cdfDataId / nil
```

Returns the effective CDF description attached to a cell. If neither a base- nor user-level CDF description is defined for the cell or its library, it returns `nil`. The resulting CDF description represents the overlay of the user-level cell CDF on the base-level cell CDF on the user-level library CDF on the base-level library CDF.

cdfGetInstCDF

```
cdfGetInstCDF(d_instId
     )
     => cdfDataId / nil
```

Returns the effective CDF description associated with an instance.

The difference between the instance's effective CDF description and the cell's effective CDF description is that the values of any CDF parameter takes into account the values of the parameters stored on the instance.

Saving Descriptions

You can save CDF descriptions using the `cdfSaveCDF` function. If this CDF description already exists, the new description is written over the old one.

cdfSaveCDF

```
cdfSaveCDF(g_cdfDataId
     )
     => t / nil
```

Saves a CDF description to disk.

The CDF description is then read in every time you open the cell of the library to which the description is attached. You can save only base-level CDF descriptions. You must have write permission on the object to which the CDF description is attached to execute this function.

Dumping and Editing Descriptions

You can save a CDF description to a file and edit it using the following functions. These functions replace `adcDump` and `cdfDumpCellCDF`.

While editing dump files remember to escape invalid characters while specifying termOrder using symbols. All symbols must be preceded by the backslash (\) to make them valid symbols in SKILL.

cdfDump

```
cdfDump(t_libName t_fileName
     [?cellName t_cellName]
     [?level s_level]
     [?edit g_edit]
     )
     => t / nil
```

Dumps the CDF description for *t_libName* and *t_cellName* into *t_fileName*. If *t_cellName* is not specified, then only the library CDF description is dumped. *t_fileName* is created in the current working directory or the directory specified with the filename. *s_level* is either `'base` or `'user`, with `'base` as the default value. If *g_edit is* `t`, a text editor window is automatically opened on *t_fileName*. The default is no editor.

Example

```
cdfDump( "nmos" "tr.mod" ?cellName "pnp" ?level 'base
    ?edit t)
```

cdfDumpAll

```
cdfDumpAll(t_libName t_fileName
     [?level s_level]
     [?edit g_edit]
     )
     => t / nil
```

Dumps the CDF description for *t_libName* and all its cells into *t_fileName*. *s_level* is either `'base` or `'user`, with `'base` as the default value. *t_fileName* is created in the current working directory or the directory specified with the filename. If *g_edit is* `t`, a text editor window is automatically opened on *t_fileName*. The default is no editor.

Example

```
cdfDumpAll("asic" "lib.mod" ?level 'base ?edit t)
```

Deleting Descriptions

You can delete a CDF description or its parameters using the following functions.

cdfDeleteCDF

```
cdfDeleteCDF(g_cdfDataId
     )
     => t / nil
```

Deletes a CDF description, including all attached parameters.

If the CDF description has been saved, the saved versions are also deleted. If this is a base-level CDF description, you must have write permission on the object to which the CDF description is attached.

cdfDeleteParam

```
cdfDeleteParam(g_cdfParamId
     )
     => t / nil
```

Deletes a CDF parameter.

If the CDF parameter is attached to a base-level CDF description, you must have write permission on the object to which the CDF description is attached.

CDF checks that no invalid parameter descriptions would result from deleting the specified parameter before it actually deletes it. This would occur if you tried to delete a base-level parameter and a user-level parameter is defined that only partially overrides the base-level description.

Copying, Finding, and Updating Data and Parameters

You can copy, find, and update CDF data and parameters using the following functions.

cdfCopyCDF

```
cdfCopyCDF(g_cellId|g_libId t_dataType g_sourceCdfDataId
     )
     => g_cdfDataId | nil
```

Copies the CDF data of the specified type from specified source to a library or cell by creating a new CDF data ID.

The destination must be specified as the ID of a library or a cell. The destination library or cell must not already have a CDF description of the specified CDF data type.

You can specify one of the following CDF data types:

- Base-level CDF data of the library (`baseLibData`)
- User-level CDF data of the library (`userLibData`)
- Base-level CDF data of the cell (`baseCellData`)
- User-level CDF data of the cell (`userCellData`)

Important Points to Note:

- If you have specified type `baseCellData` or `baseLibData`, ensure that the destination library or cell has the appropriate write permission.
- You cannot copy effective-level CDF data.

Arguments

g_cellId | g_libId	Specifies the ID of the library or cell where the CDF data must be copied.
t_dataType	Specifies the type of cdf data to be copied.
g_cdfDataId	Specifies the source of the cdf data to be copied.

Value Returned

cdfDataId	ID of the new CDF data returned if the copy operation was successful.

nil	nil returned if the copy operation failed.

Example

In the following example, base-level CDF description of nbsim in analogLib is copied to nfet in bicmos.

```
source_cell = ddGetObj( "analogLib" "nbsim" )
srcCDF = cdfGetBaseCellCDF( source_cell )
dest_cell = ddGetObj( "bicmos" "nfet" )

if( destCDF = cdfGetBaseCellCDF(dest_cell) then
             cdfDeleteCDF( destCDF)
)
destCDF = cdfCopyCDF( dest_cell "baseCellData" srcCDF )
```

cdfCopyParam

```
cdfCopyParam(g_dataId g_paramId
     )
     => g_paramId / nil
```

Copies a parameter, adding it to *dataId*.

dataId must not already have a parameter with the same name. The parameter and the dataId must not be effective objects.

cdfFindParamByName

```
cdfFindParamByName(g_cdfDataId t_name
     )
     => g_paramId / nil
```

Returns the parameter ID for the specified parameter name on the specified CDF description, if it exists. If not, it returns nil.

Use this function to search for parameters by name.

cdfUpdateInstParam

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

Stores the CDF parameters specified in the effective cell CDF of the instance master onto the specified instance. When the Id given is not for an instance or the instance master does not have CDF definition, it returns `nil`.

cdfRefreshCDF

```
cdfRefreshCDF(
     d_libId/d_cellId
     )
     => t / nil
```

Updates the CDF structure in the memory for the specified library or cell Id with the contents stored on the hard disk. Returns `nil` if the CDF structures for the specified library and cell Id is not present in memory.

If there are multiple Virtuoso sessions and you modify and save a CDF in one session, then that CDF in other sessions would be outdated. In such a case, use the `cdfRefreshCDF` function to update the CDF in other sessions.

Use this function with caution. In case multiple users modify the CDF parameters of a common component simultaneously, the saved parameter pointers may become invalid as there is no way of notifying the application after refreshing the CDF parameter values.

Arguments

`d_libId or d_cellId` Specify the *`dd_id`* of the library or the cell.

Examples

```
cdfRefreshCDF(ddGetObj("mylib" "npnpar"))
```

aedCopyCDF

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

Opens the Copy Component CDF form.

aedDeleteCDF

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

Opens the Delete Component CDF form.

Setting Scale Factors

Use the following Cadence SKILL language commands to determine the current scale factors or to set scale factors.

cdfGetUnitScaleFactor

```
cdfGetUnitScaleFactor(
     t_unitName
     )
     => t_scaleFactor
```

Description

Displays the current scale factor for the specified unit.

Arguments

`t_unitName`	The unit name for which you want to display the scale factor.

Value Returned

`t_scaleFactor`	The current scale factor for the specified unit name.

Example

Following command returns the scale factors for *power*:

```
cdfGetUnitScaleFactor("power")
```

cdfSetUnitScaleFactor

```
cdfSetUnitScaleFactor(
     t_unitName
     t_scaleFactor
     )
     => t / nil
```

Description

Sets the scale factor for the specified unit.

Arguments

`t_unitName`	The unit name for which you want to set the scale factor.
`t_scaleFactor`	The scale factor for the specified unit name.

Example

Following command sets the lengthMetric to m (millimeters):

```
cdfSetUnitScaleFactor("lengthMetric" "m")
```

cdfEditScaleFactors

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

Displays the Units Scaling Factors form which can be used to set scaling factors for displaying CDF parameters.

Other SKILL Functions

cdfParseFloatString

```
cdfParseFloatString(
     t_string
     )
     => nil / d_value / t_string
```

Description

This function uses the standard `strtod` (string to double) function to parse the input string. When the input string contains trailing non-numerical characters, the fragment of the string is compared against a supported set of scale factor designators.

For information on scale factors, see *NFET Examples*.

Value Returned

nil	when the input string cannot be parsed as a float value or without a valid scale factor, as shown below: `cdfParseFloatString("1g") => nil`

`d_value` (a float value)	when the input string can be parsed as a float value with or without a valid scale factor, as shown below:
	`cdfParseFloatString("1.0") => 1.0`
	`cdfParseFloatString("1.0u") => 1e-06`

the given string	when the input string does not contain a valid numerical representation for a float value. For example, the input string starting with a non-digit character as shown below: `cdfParseFloatString("abcd") => "abcd"`

cdfFormatFloatString

```
cdfFormatFloatString(
     t_string
     t_scaleFactor
     )
     => nil / t_val
```

Description

This function formats the input string into a value representation, if possible. It formats the input string using the input scale factor, re-converts the value to a string, and then returns the formatted string value. If the input string cannot be converted, the input string is returned with no change to it.

Arguments

`t_string`	a string representing a float value
`t_scaleFactor`	a string representing a scale factor

Value Returned

nil	if the `t_scaleFactor` given is invalid
`t_val`	when the input string can be formatted using the input scale factor. Else, the input string is returned without any change to it.

Example

```
cdfFormatFloatString("123.4" "m") => "123400.0m"
cdfFormatFloatString("10000" "M") => "0.01M"
```

cdfSyncInstParamValue

```
cdfSyncInstParamValue(
     d_instId1
     d_instId2
     )
     => t / nil
```

Description

This function generates all the CDF parameters for the first instance (*instId1*) and updates the second instance (*instId2*) with the same values. Both the instances must share the same cell.

cdfUpdateInstSingleParam

```
cdfUpdateInstSingleParam(
     d_instId
     t_paramName
     )
     => t / nil
```

Description

This function copies the specified parameter's (*t_paramName*) effective value to the specified instance (*d_instId*).

Invoking the Edit CDF Form

You have the option of modifying how the Edit CDF form opens. You can use `aedEditCDF` to specify initial library, cell, and type values.

aedEditCDF

```
aedEditCDF(
     [?libName t_libraryName]
     [?cellName t_cellName]
     [?cdfType t_cdfType]
     )
     => t
```

Description

Opens the Edit CDF form to the library, cell, and CDF type specified by *libraryName*, *cellName*, and *cdfType*.
libraryName and *cellName* must be strings referring to an existing library or cell, and *cdfType* must be `'effective'`, `'base'`, or `'user'`.

cdfGetCustomViaCDF

```
cdfGetCustomViaCDF(d_customViaId)
     => cdfDataId / nil
```

Description

Returns the effective CDF description associated with a customVia or returns nil. When the customVia's cell or library has a base or user-level CDF defined, it returns the *cdfDataId*, otherwise returns `nil`.

cdfUpdateCustomViaParam

```
cdfUpdateCustomViaParam(d_customViaId)
     => t/ nil
```

Description

Stores the parameters specified in the effective cell CDF of the customVia in the specified customVia instance. When the specified ID is not for a customVia instance or the instance master does not have CDF definition, it returns `nil`.