cadence®

Virtuoso Parasitic Aware Design SKILL Reference

Product Version ICADVM20.1
October 2020

Printed in the United States of America.

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

Preface

This user guide describes the SKILL functions that you can use with the Cadence® Virtuoso® Parasitic Aware Design flow in Cadence® IC6.1 release, and beyond. The Parasitic Aware Design flow is available in:

- The Virtuoso® Analog Design Environment (ADE L/XL/GXL)
- The Virtuoso® Schematic Editor (VSE L/XL) applications

For more information, see the *Virtuoso Parasitic Aware Design User Guide*.

This preface contains the following topics:

Scope

Unless otherwise noted, the functionality described in this guide can be used in both mature node (for example, IC6.1.8) and advanced node (for example, ICADVM18.1) releases.

Label	Meaning
(ICADVM18.1 Only)	Features supported only in ICADVM18.1 advanced nodes and advanced methodologies releases.

(IC6.1.8 Only)	Features supported only in mature node releases.

Licensing Requirements

To run Virtuoso Parasitic Aware Design from Schematic L, Schematic XL, or Analog Design Environment (ADE) L, you need either an ADE L (95200) or an ADE XL (95210) license. If an ADE L license is available, it is checked out, if not already checked out. If an ADE L license is not found, an ADE XL license is checked out.

To run Virtuoso Parasitic Aware Design from ADE XL or ADE GXL, an additional ADE GXL license token (95220) is checked out.

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

Related Documentation

What's New and KPNS

- *Virtuoso Parasitic Aware Design What's New*
- *Virtuoso Parasitic Aware Design Known Problems and Solutions*

Installation, Environment, and Infrastructure

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

Virtuoso Tools

- *Virtuoso Parasitic Aware Design User Guide*
- *Cadence Hierarchy Editor User Guide*
- *Component Description Format User Guide*

- Cadence PVS Developers Guide
- Cadence Physical Verification User Guide
- *QRC Extraction User Guide*
- *Incremental Technology Databases and Display Resources User Guide*
- *Virtuoso Analog Design Environment L User Guide*
- *Virtuoso Analog Design Environment XL User Guide*
- *Virtuoso Analog Design Environment GXL User Guide*
- *Virtuoso Design Environment User Guide*
- *Virtuoso Schematic Editor User Guide*
- *Virtuoso Spectre Circuit Simulator and Accelerated Parallel Simulator User Guide*
- *Virtuoso UltraSim Simulator User Guide*
- *Virtuoso Unified Custom Constraints User Guide*

Additional Learning Resources

Video Library

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

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

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

Virtuoso Videos Book

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

Rapid Adoption Kits

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

In addition, Cadence offers the following training courses on the Virtuoso Electrically Aware Design flow:

- Virtuoso Analog Design Environment
- Virtuoso Simulation for Advanced Nodes

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

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

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

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

Help and Support Facilities

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

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

  The Welcome Page is displayed by default when you open Cadence Help in standalone mode from a Virtuoso installation. You can also access it at any time by selecting *Help – Virtuoso Documentation Library* from any application window, or by clicking the

Home button on the Cadence Help toolbar (provided you have not set a custom home page).

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

Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

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

- Log on to Cadence Online Support

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

Feedback about Documentation

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

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

You can also submit feedback by using the following methods:

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

Understanding Cadence SKILL

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

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

Using SKILL Code Examples

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

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

Sample SKILL Code

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

axlGetRunStatus

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

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

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

Example

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

Return value

Value of the session name argument

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

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

Accessing API Help

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

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

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

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

Typographic and Syntax Conventions

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

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

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

Identifiers Used to Denote Data Types

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

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

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

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

1

Parasitic Aware Design SKILL Commands

This chapter describes the SKILL commands used in conjunction with Virtuoso Parasitic Aware Design:

Input and Output Parameters for Parasitics SKILL API

cache_id

A `cached_id` is returned by parCacheFind or parCacheGet, for a SKILL list that contains library, cell, and view names.

design_object_name

A `design_object_name` is a legal design object name in the CDBA name space.

design_object_type

A `design_object_type` is a symbol that describes the database object type of a design object. One of `'inst`, `'instTerm`, `'master`, or `'net`.

member_list

A `member_list` is a list of members where each member is a list in the form of `(design_object_name design_object_type [parameter_list])`.

parameter_list

A `parameter_list` is a list of parameters where each parameter is a list in the form of `(name_string [parameter_type] parameter_value)`. Legal values for the optional `parameter_type` are: `'int`, `'float`, `'string`, `'intrange`, `'floatrange`, `'enum`, `'enumset` and `'stringset`.

aelDisplayOPParam

```
aelDisplayOPParam(
     instName
     simParam
     [labelParam]
     [resName]
     )
     => string_list | nil
```

Description

The `aelDisplayOPParam` function returns a string list whose elements are the `simParam` of each of the instances being processed.

The instances being processed depend on the given `instName`. The function creates a list with all of the instances being considered. The instance may be a schematic instance (the result of `inst()`), or an extracted instance (for example "`/I0/M0_1_qrc`").

If a schematic instance is given in out-of-context, then the mapped extracted instances are considered, for example if `inst()` is given, the instances considered could be (`"/I0/M0" "/I0/M0_1_qrc" "/I0/M0_2_qrc" "/I0/M0_2_qrc" "I0/M0_3_qrc" "I0/M0_4_qrc"`).

Once the list is created, the `param` specified for each instance is added to the return list. This `simParam` can be any of the simulation parameters and, if not specified, the default is `id`.

Arguments

instName	String that can be a schematic instance. The result of the method `inst()`, or an extracted instance name.
simParam	Can be any simulation parameter, for example `id`.
labelParam	Optional parameter required when the name of the label parameter defined by `opParamExprList` is different than the simulation parameter being processed. For example, if the label parameter is `mFactorF` and the simulation parameter being processed is `id`, then `labelParam` must be given with the value `mFactorF`.

resName	Optional string parameter used to select the type of results from a particular analysis, for example `dcOpInfo-info`. The type of results available can be obtained using the following command: `results(?noAlias t)`. As a default, this input is set to the current type of results.

Value Returned

`string_list`	A string with a list of numbers separated by commas.
`nil`	Instance has failed to map.

Example

For an example of use and more information, see Specifying Parameters to be Displayed.

auLvsGetLabelSuffix

```
auLvsGetLabelSuffix(
     schInstanceName
     param
     )
     => suffix | nil
```

Description

The `auLvsGetLabelSuffix` command can be used to return the label suffix that is used when annotating `dcOp` or transient op points.

Arguments

schInstanceName	The schematic instance being annotated.
param	The parameter, for example `V` (voltage) and `I` (current).

Value Returned

suffix	The returned label suffix value.
`nil`	Unsuccessful.

Example

The following is an example of a mFactored transistor using custom labels:

```
auLvsGetLabelSuffix("/IO/MO" "id")
"Sum"
```

Where instance "`/IO/MO`" with parameter "`id`" returns suffix value of "`Sum`".

For more information see Backannotation of dcOp / Transient Values for M-Factor Devices on.

aelSumOPParam

```
aelSumOPParam(
     instName
     simParam
     [labelParam]
     [resName]
     )
     => integer | nil
```

Description

The `aelSumOPParam` function returns a number which is the result of adding the values of the parameters specified by `instName`.

The argument `instName` can be, for example, a schematic name which maps to multiple m-factor devices, one device, or a specific extracted name which will allow you to display specific m-factor devices values.

To do this, `aelSumOPParam` creates a list with all the instances being considered. The instance may be a schematic instance (the result of `inst()`), or an extracted instance. For example "`I0/M0_1_qrc`".

If a schematic instance is given in out-of-context, then the mapped extracted instances are considered, for example if `inst()` is given, the instances considered could be (`"/I0/M0" "/I0/M0_1_qrc" "/I0/M0_2_qrc" "/I0/M0_2_qrc" "I0/M0_3_qrc" "I0/M0_4_qrc"`). Once the list is created, the "`param`" specified for each instance is added. This "`param`" can be any of the simulation parameters, but if not specified, is "`id`" by default..

Arguments

instName	A string that can be a schematic instance. The result of the method `inst()`, or an extracted instance name.
simParam	Can be any simulation parameter, for example `id`.
labelParam	Optional parameter required when the name of the label parameter defined by `opParamExprList` is different from the simulation parameter being processed. For example, if the label parameter is `mFactorF` and the simulation parameter being processed is `id`, then `labelParam` must be given with the value `mFactorF`.

resName	Optional string parameter used to select the type of results from a particular analysis, for example `dcOpInfo-info`. The type of results available can be obtained using the following command: `results(?noAlias t)`. As a default, this input is set to the current type of results.

Value Returned

integer	A number which is the result of adding all of the simParam available in the specified `instName`.
`nil`	The instance has failed to map.

Example

For an example of use and more information, see Specifying Parameters to be Displayed.

parCacheFind

```
parCacheFind(
     t_libName
     t_cellName
     t_viewName
     )
     => cache_id | nil
```

Description

Finds an existing parasitic cache for a given design specified using library, cell and view names.

Note: The `parCacheFind` command also work with a single `dbCellViewId` argument (a library, cell, view name triplet).

Arguments

t_libName	Specify library to locate parasitic cache in.
t_cellName	Specify cell to locate parasitic cache in.
t_viewName	Specify view to locate parasitic cache in.

Value Returned

cache_id	Returns the parasitic cache if the cache has already been built.
`nil`	No cache found.

Example

```
parCacheFind( "libName" "cellName" "viewName" )
```

parCacheGet

```
parCacheGet(
     t_libName
     t_cellName
     t_viewName
     )
     => cache_id | nil
```

Description

Finds an existing parasitic cache or creates and populates the cache for a given design, specified using library, cell and view names.

Arguments

t_libName	Specify library to locate or create parasitic cache in.
t_cellName	Specify cell to locate or create parasitic cache in.
t_viewName	Specify view to locate or create parasitic cache in.

Value Returned

cache_id	Returns the parasitic cache if the the cache has already been built through a previous call to `parCacheGet`, otherwise builds the cache for the given cell view and returns it.
`nil`	No cache found.

Example

```
cache = parCacheGet( "amsPLL" "vco" "schematic" )
=> ci:0x127cfba0
```

parCacheListFilters

```
parCacheListFilters(
     d_cache
     [g_includeOutOfContext]
     )
     => filter_id_list | nil
```

Description

Lists all parasitic filters from a given cache.

Arguments

d_cache	See cache_id
[g_includeOutOfContext]	If specified and set to `t`, the out of context filters will be listed.

Value Returned

filter_id_list	A list of filter_id for all parasitic filters in the cache.
`nil`	No filters found.

Example

```
filters = parCacheListFilters( cache )
=> (ci:0x12e4d890 ci:0x12f36620)

filters~>parameters
=> ((("type" enum "R")
     ("subtype" enum "both")
     ("include" enum "all")
     ("threshold" float 0.0)
      )
      (("type" enum "C")
     ("subtype" enum "both")
     ("include" enum "all")
     ("threshold" float 0.0)
      )
  )
```

parCacheListModels

```
parCacheListModels(
     d_cache
     [g_includeOutOfContext]
     )
     => model_id_list | nil
```

Description

Lists all parasitic models from a given cache.

Arguments

d_cache	See cache_id.
[g_includeOutOfContext]	If specified and set to `t`, the out of context models will be listed.

Value Returned

model_id_list	A list of `model_id` for all parasitic models in the cache.
`nil`	No filters found.

Example

```
models = parCacheListModels( cache )
=> (ci:0x12d4f1c0 ci:0x12f82f30)

models~>type
=> (NetR NetC
```

parCachePurge

```
parCachePurge(
     d_cache
     )
     => cache_id | nil
```

Description

Purges a constraint cellview containing parasitic estimates from memory. Any changes not saved will be lost.

Caution

The constraint view may contain constraints as well as parasitic estimates and filters. Calling parCachePurge will cause any modifications to constraints as well as to parasitic objects to be lost.

Arguments

d_cache	See cache_id

Value Returned

`t`	Returns `t` when successful.
`nil`	Returns `nil` otherwise.

Example

```
cache = parCacheFind( "amsPLL" "vco" "schematic" )
=> ci:0x127cfba0
parCachePurge( cache )
=> t
```

parCacheSave

```
parCacheSave(
     d_cache
     )
     => cache_id | nil
```

Description

Saves a constraint cell view containing parasitic estimates.

Important

The constraint view may contain constraints as well as parasitic estimates and filters. Calling parCacheSave will save the constraints as well as parasitic objects.

Arguments

d_cache	See cache_id

Value Returned

`t`	Returns `t` when successful.
`nil`	Returns `nil` otherwise.

Example

```
cache = parCacheFind( "amsPLL" "vco" "schematic" )
=> ci:0x127cfba0
parCacheSave( cache )
=> t
```

parDelete

```
parDelete(
     d_parasitic_id

     )
     => t | nil
```

Description

Deletes a parasitic model or filter. After deleting the object, the `parasitic_id` will be invalid.

Note: Using the `parasitic_id` after the original object has been deleted can cause fatal errors.

Arguments

d_parasitic_id	The parasitic model or filter to be deleted.

Value Returned

`t`	Parasitic object successfully deleted.
`nil`	Parasitic object not deleted.

Example

```
parDelete( filter )
=> t
parDelete( model )
=> t
```

parFilterCreate

```
parFilterCreate(
     d_cache
     ?type S_type
     ?subtype S_subtype
     ?members l_member_list
     ?include S_include
     ?threshold f_threshold
     ?name t_name
     ?note t_note
     ?verbose g_verbose
     )
     => filter_id | nil
```

Description

Creates parasitic filters that refer to a given design object as a member.

Arguments

d_cache	See cache_id
S_type	A legal filter type: one of “R”, “C”, “L”, or “K”. The filter only applies to parasitics of the corresponding type.
S_Subtype	When *S_type* is set to “C”, *S_Subtype* indicates whether coupled or decoupled capacitance should be filtered. Legal values are “coupled”, “decoupled”, or “both”. **Note:** This argument is ignored for other filter types.
l_member_list	An ordered filter member list. See member_list.
S_include	Parasitics to include. Legal values are “all”, “none”, or “threshold”. When set to “threshold”, only parasitics with a component value greater than *f_threshold* are included.
f_threshold	A floating point value. Ignored unless *S_include* is set to “threshold”.
t_name	A string that uniquely identifies the filter in the cache. If not specified, a name will be generated automatically.
t_note	A string note to be attached to the filter.

g_verbose	A boolean argument that controls whether a message is displayed to inform of the successful creation of a filter. Defaults to `t`.

Value Returned

filter_id	The `filter_id` of the new filter.
`nil`	Filter not created.

Example

The following examples create three filters. The meaning of each filter when refining an extracted view is:

- filterC - remove all parasitic capacitance between ibias and gnd!
- filterCC - include all parasitic coupled capacitance between ibias and all other nets (excluding supply nets).
- filterR - remove all parasitic resistors below 1 ohm on the gnd! net.

```
filterC = parFilterCreate( cache ?type "C" ?members list( list( "gnd!" 'net
) list( "ibias" 'net ) ) ?include "none" )
=> ci:0x129033a0

filterCC = parFilterCreate( cache ?type "C" ?subtype "coupled" ?members
list( list( "ibias" 'net ) ) ?include "all" )
=> ci:0x12e3d7f8

filterR = parFilterCreate( cache ?type "R" ?members list( list( "gnd!" 'net
) ) ?include "threshold" ?threshold 1.0 )
=> ci:0x12f9fac8
```

parFind

```
parFind(
     d_cache
     t_name

     )
     => parasitic_id | nil
```

Description

Finds a parasitic model or filter in a given cache.

Arguments

d_cache	The parasitic cache in which the model or filter belongs (see cache_id).
t_name	The name of the parasitic model or filter to be found.

Value Returned

parasitic_id	The `parasitic_id` of the object found.
`nil`	No model or filter found.

Example

```
model = parFind( cache "Constr_5" )
=> ci:0x12f82f30

model->type
=> netC
```

parModelCreateCustom

```
parModelCreateCustom(
     d_cache
     ?net t_net
     ?simParams l_simParams
     ?simSweeps l_simSweeps
     ?parLib t_parLib
     ?parCell t_parCell
     ?parView t_parView
     ?terminalMap l_terminalMap
     ?name t_name
     ?note t_note
     ?verbose g_verbose
     )
     => model_id | nil
```

Description

Creates a new customization parasitic estimate model for selected nets. This model is defined in the cellview defined by the parLib, parCell, and parView arguments.

Arguments

d_cache	See cache_id.
t_net	Name of net to create resistance model for.
t_type	Type of estimate. Star model or stitched extracted net.
l_simParams	Simulation parameters, a list alternating between parameter name and value.
l_simSweeps	Simulation sweeps, a list alternating between parameter name and sweep value.
t_parLib	Library where the extracted view is located.
t_parCell	Cell where the extracted view is located.
t_parView	Name of the extracted view.
l_terminalMap	List of instance terminals connecting non-hierarchical instances to the net. This list is calculated automatically.
t_name	A string that uniquely identifies the model in the cache. If not specified, a name will be generated automatically.

t_note	A string note to be attached to the model.
g_verbose	A boolean argument that controls whether a message is displayed to inform of the successful creation of a model. Defaults to `t`.

Value Returned

model_id	The `model_id` of the new estimate model.
`nil`	Model not created.

Example

To create a parasitic resistance model for net `vdd!`:

```
cache = parCacheGet( “analogLib” “presister” “symbol” )
parModelCreateCustom( cache ?nets "vdd!" ?parLib "analogLib" ?parCell "presister"
?parView "symbol" )
=> ci:0x12be3998
```

parModelCreateNetC

```
parModelCreateNetC(
     d_cache
     ?net t_net
     ?type t_type
     ?extLib t_extLib
     ?extCell t_extCell
     ?extView t_extView
     ?netMap t_netMap
     ?include t_include
     ?threshold t_threshold
     ?members l_members
     ?simParams l_simParams
     ?simSweeps l_simSweeps
     ?name t_name
     ?note t_note
     ?verbose g_verbose
     )
     => model_id | nil
```

Description

Creates a new parasitic capacitance estimate between two nets.

Arguments

d_cache	See cache_id.
t_net	Name of net to create capacitance model for.
t_type	Type of estimate. Star model or stitched extracted net.
t_extLib	Library where extracted view is located.
t_extCell	Cell where extracted view is located.
t_extView	Name of the extracted view.
t_netMap	List of pair's mapping net members to nets in the extracted view.
t_include	Chose to include `all`, `none`, `threshold`, or `lump`.
t_threshold	If `include` = `theshold`, parasitics with values below the threshold will not be stitched into the estimate view.
l_members	List of two net members. See member_list.

l_simParams	Simulation parameters; specifying a name value pair for the parameter.
l_simSweeps	Simulation sweeps; specifying a name value pair for the parameter.
t_name	A string that uniquely identifies the model in the cache. If not specified, a name will be generated automatically.
t_note	A string note to be attached to the model.
g_verbose	A boolean argument that controls whether a message is displayed to inform of the successful creation of a model. Defaults to `t`.

Value Returned

model_id	The `model_id` of the new estimate model.
`nil`	Model not created.

Example

To create a `10f` parsitic capacitance estimate between nets `ibias` and `gnd!`:

```
parModelCreateNetC( cache ?members list( list( "gnd!" 'net ) list( "ibias"
'net ) ) ?simParams list( "c" "10f" ) )
=> ci:0x12fde108
```

parModelCreateNetK

```
parModelCreateNetK(
     d_cache
     ?members t_members
     ?simParams l_simParams
     ?simSweeps l_simSweeps
     ?name t_name
     ?note t_note
     ?verbose g_verbose
     )
     => model_id | nil
```

Description

Creates new parasitic mutual-inductance estimate models between the inductance of the specified instance terminals. The members specified are the instance terminals whose estimate inductance is to be considered creating mutual inductance. If the instance terminals do not have an associated estimate inductance, that will be created as a side-effect.

Arguments

d_cache	See cache_id.
t_members	List of the instance terminals whose estimate inductance is considered for creating mutual inductance. See member_list.
l_simParams	Simulation parameters, a list alternating between parameter name and value.
l_simSweeps	Simulation sweeps, a list alternating between parameter name and sweep value.
t_name	A string that uniquely identifies the model in the cache. If not specified, a name will be generated automatically.
t_note	A string note to be attached to the model.
g_verbose	A boolean argument that controls whether a message is displayed to inform of the successful creation of a model. Defaults to `t`.

Value Returned

model_id	The `model_id` of the new estimate model.
`nil`	Model not created.

Example

To create a parasitic mutual inductance model between estimate inductances of instance terminals "`MP0:D`" and "`MN0:D`", where the "`k`" value of mutual inductance is 1.

```
parModelCreateNetK( cache ?members list("/MP0:D "/MNO:D") ?simParams
list("k" "1" ))
=> ci:0x12be3998
```

parModelCreateNetL

```
parModelCreateNetL(
     d_cache
     ?net t_net
     ?type t_type
     ?extLib t_extLib
     ?extCell t_extCell
     ?extView t_extView
     ?extNet t_extNet
     ?terminalMap l_terminalMap
     ?include t_include
     ?threshold t_threshold
     ?members t_members
     ?simParams l_simParams
     ?simSweeps l_simSweeps
     ?name t_name
     ?note t_note
     ?verbose g_verbose
     )
     => model_id | nil
```

Description

Creates a new parasitic inductance estimate model for a net. The model is star-shaped with an inductance connecting members to a central node. The members are the instance terminals connecting instances to the net. Also included are the terminals of the net. The member list provides an option to specify the list of instance terminals to include. If the members list is `nil`, all instances of the net are selected.

Arguments

d_cache	See cache_id.
t_net	Name of the net to create a resistance model for.
t_type	Type of estimate. Star model or stitched extracted net.
t_extLib	Library where extracted view is located.
t_extCell	Cell where extracted view is located.
t_extView	Name of the extracted view.
t_extNet	Name of net in extracted view.

l_terminalMap	List of pairs mapping the current design's terminals to the extracted design's terminals. Both elements of the pair provide the instance and terminal name in the schematic namespace, separated by a colon.
t_include	Chose to include `all`, `none`, `threshold`, or `lump`.
t_threshold	If `include` = `theshold`, parasitics with values below the threshold will not be stitched into the estimate view.
t_members	List of the instance terminals of the net from which the inductance model is built. If the list is `nil`, all instance terminals of the net are considered. See member_list.
l_simParams	Simulation parameters, a list alternating between parameter name and value.
l_simSweeps	Simulation sweeps, a list alternating between parameter name and sweep value.
t_name	A string that uniquely identifies the model in the cache. If not specified, a name will be generated automatically.
t_note	A string note to be attached to the model.
g_verbose	A boolean argument that controls whether a message is displayed to inform of the successful creation of a model. Defaults to `t`.

Value Returned

model_id	The `model_id` of the new estimate model.
`nil`	Model not created.

Example

To create a parasitic inductance model for net `vdd!`. Each inductance will have a value of 5 henry:

```
parModelCreateNetL( cache ?net "vdd!" ?simParams list ( "l" "5") ) )
=> ci:0x12be3998
```

parModelCreateNetR

```
parModelCreateNetR(
     d_cache
     ?net t_net
     ?type t_type
     ?extLib t_extLib
     ?extCell t_extCell
     ?extView t_extView
     ?extNet t_extNet
     ?terminalMap l_terminalMap
     ?include t_include
     ?threshold t_threshold
     ?members l_members
     ?simParams l_simParams
     ?simSweeps l_simSweeps
     ?name t_name
     ?note t_note
     ?verbose g_verbose
     )
     => model_id | nil
```

Description

Creates a new parasitic resistance model.

Arguments

d_cache	See cache_id.
t_net	Name of net to create resistance model for.
t_type	Type of estimate. Star model or stitched extracted net.
t_extLib	Library where extracted view is located.
t_extCell	Cell where extracted view is located.
t_extView	Name of the extracted view.
t_extNet	Name of net in extracted view.
l_terminalMap	List of pairs mapping the current design's terminals to the extracted design's terminals. Both elements of the pair provide the instance and terminal name in the schematic namespace, separated by a colon.
t_include	Chose to include `all`, `none`, `threshold`, or `lump`.

t_threshold	If `include` = `theshold`, parasitics with values below the threshold will not be stitched into the estimate view.
l_members	List of instance terminals on the net. See member_list.
l_simParams	Simulation parameters; specifying a name value pair for the parameter.
l_simSweeps	Simulation sweeps; specifying a name value pair for the parameter.
t_name	A string that uniquely identifies the model in the cache. If not specified, a name will be generated automatically.
t_note	A string note to be attached to the model.
g_verbose	A boolean argument that controls whether a message is displayed to inform of the successful creation of a model. Defaults to `t`.

Value Returned

model_id	The `model_id` of the new estimate model.
`nil`	Model not created.

Example

To create a parasitic resistance model for net `vdd!`.

```
parModelCreateNetR( cache ?net "vdd!" ?simParams list( "r" "5" ) )
=> ci:0x12be3998
```

Note: Each resistor will be 5 ohms.

parModelListSimParams

```
parModelListSimParams(
     d_model_id
     )
     => sim_param_list | nil
```

Description

Lists the suimulation parameters associated with a parasitic estimate. These are the parameters that will be set on the parasitic model that is inserted into the netlist.

Arguments

d_model_id	The parasitic model whose simulation parameters you want to list.

Value Returned

sim_param_list	List of simulation parameter names and values. The list alternates between parameter names and values.
nil	No simulation parameters have been set on the estimate.

Example

```
parModelListSimParams( model )
=> ("r" "5")
```

parModelListSimSweeps

```
parModelListSimSweeps(
     d_model_id
     )
     => sim_sweep_list | nil
```

Description

Lists the simulation sweeps associated with a parasitic estimate.

Arguments

d_model_id	The parasitic model whose simulation sweeps you want to list.

Value Returned

sim_sweep_list	List of simulation parameter names and sweeps. The list alternates between parameter name and sweep.
nil	No simulation sweeps have been set on the estimate.

Example

```
parModelListSimSweeps( model )
=> ("r" "1:2:5")
```

parModelUpdateSimParams

```
parModelUpdateSimParams(
     d_model_id
     l_sim_param_list
     )
     => t | nil
```

Description

Updates the values of the listed simulation parameters.

Arguments

d_model_id	The parasitic model whose simulation parameters you want to update.
l_sim_param_list	The list of parameters to be updated. This should be a list that alternates between parameter name and value.

Value Returned

`t`	Parameter values successfully updated.
`nil`	Parameter values not updated.

Example

To set the component values for a parasitic resistance model to 1 ohm:

```
parModelUpdateSimParams( model list( "r" "1" ) )
=> t
```

parModelUpdateSimSweeps

```
parModelUpdateSimSweeps(
     d_model_id
     l_sim_sweep_list
     )
     => t | nil
```

Description

Updates the sweeps of the listed simulation parameters.

Arguments

d_model_id	The parasitic model whose simulation sweeps you want to update.
l_sim_sweep_list	The list of sweeps to be updated. This should be a list that alternates between name and sweep value.

Value Returned

`t`	Sweeps successfully updated.
`nil`	Sweeps not updated.

Example

To set the components on a parasitic resistance estimate to sweep from 1 to 5 ohms in steps of 2 ohms:

```
parModelUpdateSimSweeps( model list( "r" "1:2:5" ) )
=> t
```

parObjectListFilters

```
parObjectListFilters(
     d_cache
     t_design_object_name
     t_design_object_type
     )
     => filter_id_list | nil
```

Description

Lists all parasitic filters that refer to a given design object as a member.

Arguments

d_cache	See cache_id
t_design_object_name	See design_object_name
t_design_object_type	See design_object_type

Value Returned

filter_id_list	A list of filter_id for all filters that refer to the named design object as a member.
nil	No filters found.

Example

```
filters = parObjectListFilters( cache "gnd!" 'net )
=> (ci:0x129033a0 ci:0x12f9fac8)
filters~>parameters
=> ((("type" enum "C")
     ("subtype" enum "both")
     ("include" enum "none")
     ("threshold" float 0.0)
       )
       (("type" enum "R")
     ("subtype" enum "both")
     ("include" enum "threshold")
     ("threshold" float 0.0)
```

```
        )
    )
```

parObjectListModels

```
parObjectListModels(
     d_cache
     t_design_object_name
     t_design_object_type
     )
     => model_id_list | nil
```

Description

Lists all parasitic models that refer to a given design object as a member.

Arguments

d_cache	See cache_id.
t_design_object_name	See design_object_name.
t_design_object_type	See design_object_type.

Value Returned

model_id_list	A list of `model_id` for all models that refer to the named design object as a member.
`nil`	No models found.

Example

```
models = parObjectListModels( cache "gnd!" 'net )
=> (ci:0x131973e0 ci:0x131a4e28 ci:0x131b7528 ci:0x12f82f30)

models~>type
=> (netC netC netC netC)
```

parRemoveMembers

```
parRemoveMembers(
     d_parasitic_id
     l_member_index_list

     )
     => t | nil
```

Description

Removes members from a parasitic model or filter.

Arguments

d_parasitic_id	The parasitic model or filter to remove members from.
l_member_index_list	List of integer indexes indicating the members to remove (for example (2 4) for 2nd or 4th member)

Value Returned

t	Members successfullly removed.
nil	Members not removed.

Example

To remove two instance terminals from a parasitic resistance estimate (as a result no resistors will be inserted for these members and they will be connected directly to the center of the star network):

```
parRemoveMembers( model list( list( "D2:MINUS" 'instTerm ) list( "D3:MINUS"
'instTerm ) ) )
=> t
```

parResetAllParams

```
parResetAllParams(
     d_parasitic_id

     )
     => t | nil
```

Description

Resets all model or filter parameters to default values.

Arguments

d_parasitic_id	The parasitic model or filter id whose parameter values you want to reset.

Value Returned

t	Parameter values successfully reset to default values.
nil	Parameter values not reset.

Example

```
parResetAllParams( filter )
=> t
```

parResetParams

```
parResetParams(
     d_parasitic_id
     l_parameter_name_list

     )
     => t | nil
```

Description

Resets given model or filter parameters to default values.

Arguments

d_parasitic_id	The parasitic model or filter id whose parameter values you want to reset.
l_parameter_name_list	List of parameter names to be reset to their default values.

Value Returned

`t`	Parameter values successfully reset to default values.
`nil`	Parameter values not reset.

Example

```
parResetParams( filterR list( "include" "threshold" ) )
=> t
```

parSetNote

```
parSetNote(
     d_parasitic_id
     t_note_string

     )
     => t | nil
```

Description

Replaces the note on a parasitic model or filter.

Arguments

d_parasitic_id	The parasitic model or filter id whose note you want to replace.
t_note_string	The new note string.

Value Returned

`t`	Note successfully replaced.
`nil`	Note not replaced.

Example

```
parSetNote( filter "Filter out all C between gnd! and ibias" )
=> t
```

parUpdateMembers

```
parUpdateMembers(
     d_parasitic_id
     l_member_list

     )
     => t | nil
```

Description

Updates parasitic model or filter members and their parameters. Existing members will have their parameter list updated and new members will be added at the end.

Note: This function does not reorder existing members. Reordering should be done in conjunction with `parRemoveMembers`.

Arguments

d_parasitic_id	The parasitic model or filter id whose members and parameters you want to update.
l_member_list	The list of members to be updated (see member_list).

Value Returned

`t`	Members and parameters successfully updated.
`nil`	Members and parameters not updated.

Example

To update a parasitic capacitance filter so that it applies to all nets contained under a hierarchical instance:

```
parUpdateMembers( filterC list( list( "I15" 'inst ) ) )
=> t
```

parUpdateParams

```
parUpdateParams(
     d_parasitic_id
     l_parameter_list

     )
     => t | nil
```

Description

Updates the parameter values of the listed parameters. Default values will reset the parameter to default and the storage for the default value will be deleted. Enumerated values will be reset first, then updated rather than appened.

Arguments

d_parasitic_id	The parasitic model or filter id whose parameter values you want to update.
l_parameter_list	The list of parameters to be updated (see parameter_list).

Value Returned

`t`	Parameter values successfully updated.
`nil`	Parameter values not updated.

Example

To update a parasitic resistance filter to exclude all resistors with a value less than 1.0 ohm:

```
parUpdateParams( filterR list( list( "include" "threshold" ) list("threshold" 1.0
) ) )
=> t
```

mspsMapNetName

```
mspsMapNetName(
     h_hdbConfigId
     t_name
     )
     => t_frag_name | nil
```

Description

Maps a hierarchical schematic net name to a fragment of the equivalent net in the extracted view.

This function is required to be used only with Virtuoso executable and not with OCEAN executable. Prior to IC6.1.4, it was required to use this function to map schematic net names for OCEAN and ADE data access function calls while simulating an extracted view using a configuration. From IC6.1.4 onwards, OCEAN and ADE data access functions automatically map schematic names into the simulated extracted view and it is no longer necessary to call this function.

Arguments

h_hdbConfigId	Configuration cellview identifier returned by hdbOpen.
t_name	Hierarchical schematic net name

Value Returned

`t_frag_name`	Hierarchical name of a net fragment in the extracted view that maps to the net name in the schematic view. This can be any one of the net fragments in the extracted view that map to schematic net. If none of the instances in the hierarchical schematic name is bound to an extracted view, the schematic name is returned unmodified. In this case the supplied hierarchical name does not lead into an extracted view. For example, if you have the name `/I0/I1/I2/netA` it means that none of I0, I1, I2 have been bound to an extracted view in the config. In this case, the name is returned unmodified as it does not need to be mapped into an extracted view.

`nil`	Returns `nil` if the schematic name does not match the extracted view. In this case, the supplied name does lead into an extracted view, but the net name cannot be mapped into that view because, for example, a wrong net name is given. For example, if you bind I1 to an extracted view in the previous example, the name `/I0/I1/I2/netA` needs to be mapped now. But, if netA does not exist, then the function returns `nil`.

Example

In these examples, instance /X1 is bound to an extracted view and instance /XA to the schematic view.

```
cfg = hdbOpen("worklib" "TB1_vco_RCXcompare" "config" "r") =>
hdbcConfigType:0x0xbe9d948
```

```
mspsMapNetName( cfg "/X1/I15/n3" )
=> "/X1/2:I15|n3"
```

Here, net /X1/I15|n3 in the schematic maps to net /X1/2:I15/n3 in the extracted view.

```
mspsMapNetName( cfg "/XA/I15/n3" )
=> "/XA/I15/n3"
```

Here, instance XA is bound to the schematic view. Therefore, the name does not need to be mapped.

```
mspsMapNetName( cfg "/X1/I15/inx2" )
=> nil
```

Here, inx2 ia not a valid net name in the design.

axlGetParasiticViewName

```
axlGetParasiticViewName(
     t_sessionName
     t_flowName
     )
     => t_viewName
```

Description

Gets the name of the parasitic view set for the given flow in ADE XL.

Arguments

t_sessionName	Name of the ADE XL session
t_flowName	Name of the flow for which you need to get the name of the parasitic view **Valid Values:** `Estimated`, `Extracted`, or `Layout`

Value Returned

t_viewName	Name of the parasitic view that is set to be used in the given flow.

Example

The following example shows how to get the view names by using this function.

```
session = axlGetWindowSession()
=> "session0"
axlGetParasiticViewName("session0" 'Extracted)
=> "av_extracted_rc"
axlGetParasiticViewName("session0" 'Estimated)
=> "estimated"
axlGetParasiticViewName("session0" 'Layout)
=> "netlist_layout"
```

Related Function

axlSetParasiticViewName

axlMapInstTermToNet

```
axlMapInstTermToNet(
     t_instPathName
     t_termName
     [t_dataDir]
     [g_verbose]
     )
     => t_netName
```

Description

This function is useful while doing out-of-context probing with a config view. It maps an instance terminal to its corresponding net connection in the configured view, which can be a schematic, a parasitic/LDE, or an extracted view. Instead of directly using net names in calculator expressions (in an ADE output), you can call `axlMapInstTermToNet` from within the expression to dynamically return the name of the net connected to an instance terminal. In this case, even if the configured view is modified and the net connected to a terminal is changed, the calculator function can get the correct net name connected to the given instance terminal.

Arguments

t_instPathName	Path to the instance terminal in the schematic design hierarchy
t_termName	Name of the instance terminal
t_dataDir	(Optional) Path to the results directory.
g_verbose	(Optional) Sets the verbose mode on or off. When set to t, the function prints log details with design name, extracted cellview name, and the extracted net name.

Value Returned

t_netName	Net name in the extracted view to which the instance terminal is mapped.

Example

In this example, a parasitic RC extracted simulation is run in ADE. In the output, the calculator function, VT, uses axlMapInstTermToNet to use the net name mapped to the instance terminal `I3/MP0:D`.

```
VT(axlMapInstTermToNet( "/I1/I2/I3/MP0" "D" ) )
```

In this example, the axlMapInstTermToNet function internally returns the mapped net name as
`"/I1/14:I2|I3|net1"`

axlSetParasiticViewName

```
axlSetParasiticViewName(
     t_sessionName
     t_flowName
     t_viewName
     )
     => t_viewName
```

Description

Sets the name of the parasitic view to be used for the given flow in ADE XL.

Arguments

t_sessionName	Name of the ADE XL session
t_flowName	Name of the flow for which you need to set a parasitic view **Valid Values:** `Estimated`, `Extracted`, or `Layout`
t_viewName	Name of the parasitic view to be used in the given flow.

Value Returned

t_flowName	Name of the flow for which the view name has been set.

Example

The following example shows how to set a view name to be used for the extracted flow.

```
session = axlGetWindowSession()
=> "session0"
axlSetParasiticViewName("session0" 'Extracted "av_extracted_rc")
=> (Extracted)
```

Related Function

axlGetParasiticViewName