Virtuoso Unified Custom Constraints SKILL Reference

Product Version ICADVM20.1
October 2020

Contents

Preface

The Cadence® Virtuoso® unified custom constraint management system allows you to establish design needs, save them as constraints, and share those constraints across specification, simulation, and implementation to drive the accelerated layout solution with reduced errors. A constraint-driven design preserves the design intent by enabling efficient design collaboration. For more information, see the *Virtuoso Unified Custom Constraints User Guide*.

This guide describes the SKILL functions that you can use with this unified custom constraint management system. You can use these SKILL functions to modify the constraints to suit your needs.

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

- The Virtuoso design environment and application infrastructure mechanisms designed to support consistent operations between all Cadence® tools.
- The applications used to design and develop integrated circuits in the Virtuoso design environment, notably, the Virtuoso Layout Suite, and Virtuoso Schematic Editor.
- The Virtuoso design environment technology file.
- Component description format (CDF), which lets you create and describe your own components for use with Layout XL.

This preface contains the following topics:

- Identifiers Used to Denote Data Types
- Orientation Abbreviations

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, ICADVM20.1) releases.

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

Licensing Requirements

The unified custom constraint management system is available in:

- Virtuoso® Schematic Editor XL (Schematics XL)
- Virtuoso® Layout Suite XL (Layout XL)
- Virtuoso® Layout Suite GXL (Layout GXL) **(IC6.1.8 Only)**

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

Related Documentation

What's New and KPNS

- *Virtuoso Unified Custom Constraints What's New*
- *Virtuoso Unified Custom Constraints Known Problems and Solutions*

Installation, Environment, and Infrastructure

- *Cadence Installation Guide*

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

Technology Information

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

Virtuoso Tools

IC6.1.8 Only

- *Virtuoso Layout Suite XL User Guide*
- *Virtuoso Analog Placer User Guide, Using the Cell Planner* chapter

ICADVM 20.1 Only

- *Virtuoso Layout Suite: Basic Editing*
- *Virtuoso Layout Suite XL: Connectivity Driven Editing*

IC6.1.8 and ICADVM 20.1

- *Virtuoso Unified Custom Constraints Configuration Guide*
- *Virtuoso Unified Custom Constraints Getting Started Guide*
- *Virtuoso Unified Custom Constraints User Guide*
- *Virtuoso Schematic Editor User Guide*
- *Virtuoso Module Generator 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 the related 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 Virtuoso unified custom constraints functionality and related Virtuoso tools:

- Using Virtuoso Constraints Effectively
- Virtuoso Schematic Editor
- Virtuoso Layout Design Basics
- Virtuoso Layout Pro: T3 Basic Commands (XL)
- Virtuoso Layout Pro: T4 Advanced Commands (XL)
- Virtuoso Connectivity-Driven Layout Transition
- Virtuoso Layout 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 as presented in the manual. 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	nmpII 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*.

Orientation Abbreviations

The following orientation abbreviations are used in this manual:

Abbreviation	Description
R0	no orientation
R90	rotates 90 degrees
R180	rotates 180 degrees
R270	rotates 270 degrees
MX	mirrors about the X axis

Abbreviation	**Description**
MXR90	mirrors about the X axis 90 degrees
MY	mirrors about the Y axis
MYR90	mirrors about the Y axis 90 degrees

1

Custom Constraints Functions

This chapter covers details about the SKILL functions available for the Constraints Manager and Circuit Prospector assistants. It contains sections on the following topics:

Note: For information on creating SKILL Constraint User Type (SCUT) objects using Cadence® SKILL language (instead of using the graphical user interface), see CST Access SKILL Functions

Constraint Integration and Third Party Tools

Any non-Cadence tools, that are integrated into DFII, can also register their proprietary constraints within the Constraint system using a `config.xml` file as discussed in the Creating Custom Constraint Types chapter of the *Virtuoso Unified Custom Constraints Configuration Guide*.

For more information on this feature, contact your local Cadence representative.

Note: Access to this facility will incur an additional cost.

Constraint Cache

The constraint cache is a collection of constraint data that is built on demand for a given design and contains all the constraints related to that design.

There are two distinct APIs, ciCacheFind which only returns an already created cache, and ciCacheGet which will either return an already created cache or force one to be built.

Note: Currently, APIs are not supported on a cellview. In future, support will be provided for text designs that do not have a cellview, and the flexibility provided by `ciCacheFind` and `ciCacheGet` will result in some applications wanting to use `ciCacheFind` and defer all constraint API usage when a cache has not been built by another application. Other applications may only use `ciCacheGet`.

Input and Output Parameters for Custom Constraints Functions

cache

Is defined as: `u_cache | (libName cellName viewName)`

u_cache

Is a SKILL user type object that references a constraint cache.

u_constraint

Is a SKILL user type object that references a constraint.

parameter_value

A parameter value is a SKILL atom or list that contains a legal value for the `parameter`.

- Enumerated parameters and arrays are expressed as homogeneous value type lists.
- Boolean values are evaluated to SKILL `t/nil` for true/false respectively (and `"TRUE"/"FALSE"` strings both evaluate to `t`, while `0/1` evaluate to `t/nil` respectively).

When `parameter_type` is specified in a parameter, the atom members are converted to the specified values using standard arithmetic conversion.

Strings are not converted to and from numerals. For example, "1" is not converted to the integer 1 and the integer 1 is not converted to the string "1".

parameter

Is defined as: `(name_string [parameter_type] parameter_value)`

A constraint parameter is a list that contains the parameter name and `parameter_value` and, optionally, a `parameter_type`.

parameter_list

Is defined as a list of constraint parameters:

```
(constraint_parameter1 constraint_parameter2 ...)
```

constraint_member

Is defined as: `(design_object_name design_object_type [parameter_list ])`

A constraint member is a list containing the member name and type and, optionally, a list of member `parameter`. Constraints always refer to constraint members using a name and type pair to avoid conflicts in design namespaces between design objects: the pair (name type) creates a unique namespace for any design hierarchy.

design_object

Is defined as: `(design_object_name design_object_type )`

- A design object is similar to a `constraint_member`, in that it is always a `design_object_name design_object_type` pair and it cannot have parameters.
- A design object is part of the constraint `cache` if any constraint has been set on that design object.
- A design object contains references to all constraints that refer to it as a member.

design_object_name

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

member_list

Is defined as a list of constraint members:

```
(constraint_member1 constraint_member2 ...)
```

design_object_list

Is defined as: `member_list` `constraint_member`

A design object list is a list of design objects.

Pre-Defined Symbols

The following is a list of constraint pre-defined symbols:

design_object_type

Is defined as: `inst | net | modgen | netClass | cluster | pin | instTerm | master | terminal | boundary | netClassGroup | symmetry | diffPair | matchedLength | bus`

A design object type is a symbol that describes the database object type of a design object.

The design objects type, such as `inst`, `net`, `pin`, `instTerm`, `terminal`, and `boundary` are used when specifying those types of design object as members of a constraint.

```
ciConCreate(cache 'diffPair ?members list( list("N1" 'net) list("N2" 'net)))
```

It specifies two nets `N1` and `N2` as members of a Diff Pair constraint.

The `modgen`, `netClass`, `cluster`, `netClassGroup`, `symmetry`, `diffPair`, `matchedLength`, and `bus` are used when specifying those type of constraints as members of another constraints. These constraints can also be members of other constraints. For example, Symmetry (on Modgens), Guardring (on Clusters) or to create hierarchical relationships like a Cluster comprising of other Clusters.

```
ciConCreate(cache 'symmetry ?members list( list("Moden_1" 'modgen)))
```

It specifies Modgen constraint `Modgen_1` as a member of a Symmetry constraint.

```
ciConCreate(cache 'cluster ?members list( list("I0" 'inst) list("Cluster_2"
'cluster)))
```

It specifies instance `I0` and cluster `Cluster_2` as members of a Cluster constraint.

parameter_type

Is defined as: `int | float | string | intrange | floatrange | enum | enumset | stringset`

constraint_type

A list of default constraint types corresponding to SKILL names is illustrated below:

Constraint Type	**SKILL Name**
Alignment (*Placement*)	alignment
Area Utilization	areaUtilization
Area Pin Group Guide (*Placement*)	areaPinGroupGuide
Bus (*Routing*)	bus
Cell Boundary (*Placement*)	cellBoundary
Cluster (*Placement*)	cluster
Cluster Boundary (*Placement*)	clusterBoundaryDef
Correlation (*Electrical*)	correlation
Current	current
Diff Pair (*Routing*)	diffPair
Disable Permutation (*Placement*)	disablePermutation
Distance (*Placement*)	distance
Edge Pin Group Guide (*Placement*)	edgePinGroupGuide
Fixed (*Routing* / *Placement*)	fixed
Guard Ring (*Placement*)	powerStructure
High Precision C Extraction (*Electrical*)	highPrecisionExtraction
High Precision R Extraction (*Electrical*)	highPrecisionResExtraction
IRDrop	IRDrop
Locked (*Routing* / *Placement*)	locked
Matched Capacitance (*Electrical*)	matchedCapacitance
Matched LDE Parameters	matchedLDE
Matched Length (*Routing*)	matchedLength
Matched Parameters	matchedParameters
Matched Orientation (*Placement*)	relativeOrientation
Max Capacitance (*Electrical*)	maxCapacitance

Constraint Type	SKILL Name
Max Coupling Capacitance (*Electrical*)	maxCouplingCapacitance
Max Resistance (*Electrical*)	maxResistance
Modgen (*Placement*)	modgen
Net Class (*Routing*)	netClass
Net Class Group (*Routing*)	netClassGroup
Net Class Hier Group (*Routing*)	netClassHierGroup
Net Priority (*Routing*)	priority
Orientation (*Placement*)	orientation
Placement Path (*Placement*)	placementPath
Shielding (*Routing*)	shielding
Symmetry (*Routing* / *Placement*)	symmetry
Process Rule Override (*Routing*)	processOverride
Rail (*Placement*)	rail
Voltage	voltage

For information on the current list of *Constraint Manager* constraints, see Default Constraint Types in the *Virtuoso Unified Custom Constraints User Guide*.

parameter_key

Is defined as: `?params`

member_key

Is defined as: `?members`

name_key

Is defined as: `?name`

axis_key

Is defined as: `?axis`

axis

Is defined as: `axis_name_string [parameter_list]`

Constraint Manager Assistant Customization SKILL Commands

The following SKILL commands are available for customizing the *Constraint Manager* assistant:

ciA...	
ciAddHierarchicalNotes	ciAxisCreate
ciAddLeadingSlash	ciAxisDelete
ciAddProcessRules	ciAxisExists
ciAddRuleGroup	ciAxisListCon
ciAddTrailingSlash	ciAxisListParams
ciAllCellViewsInHierarchy	ciAxisReplaceParams
ciC...	
ciCacheCallbackRegister	ciConGetAxisName
ciCacheCallbackUnregister	ciConGetCache
ciCacheCallbackUpdate	ciConGetComment
ciCacheCellName	ciConGetCreatedTime
ciCacheConstraintCellName	ciConGetMembersOfType
ciCacheConstraintLibName	ciConGetName
ciCacheConstraintViewName	ciConGetNote
ciCacheDiscardEdits	ciConGetOwner
ciCacheFind	ciConGetPriority
ciCacheGet	ciConGetStatus
ciCacheGetAllNetNames	ciConGetType
ciCacheGetCellView	ciConIsInContext
ciCacheGetEnabledNotifications	ciConIsOutOfContext
ciCacheIsLayout	ciConCreate
ciCacheIsModified	ciConCreateExpanded
ciCacheIsWritable	ciConIsOverridden

ciCacheLCV	ciConIsWritable
ciCacheLibName	ciConListMembers
ciCacheListAxesNames	ciConListMemberNames
ciCacheListCon	ciConListParams
ciCacheListConstrainedObjects	ciConListParamNames
ciCacheListConstrainedObjectNames	ciConListTemplates
ciCacheListTemplates	ciConp
ciCacheListTypeNames	ciConRegisterCallback
ciCacheListTypes	ciConRemoveMembers
ciCacheMakeEditable	ciConResetAllParams
ciCacheMakeReadOnly	ciConResetParams
ciCacheNeedRefresh	ciConSetAxis
ciCacheNotifications	ciConSetNote
ciCachep	ciConSetPriority
ciCachePurge	ciConSetStatus
ciCacheSave	ciConstraintsForType
ciCacheTopCellName	ciConstraintLCV
ciCacheTopLibName	ciConstraintViewLessp
ciCacheTopViewName	ciConTypeHasNamedParameter
ciCacheTransfer	ciConUnregisterCallback
ciCacheTransferSelection	ciConUpdateCallback
ciCacheViewName	ciConUpdateMemberParams
ciCheckConstraints	ciConUpdateMembers
ciCombineInstNetsPins	ciConUpdateParams
ciConAppendOneMember	ciConUprevCellBoundary
ciConBaseName	ciConVerify
ciConCallbackIsRegistered	ciConvertNestedNetClassToNetClassHierGroup
ciConDelete	ciCreateFilter
ciConFind	ciCurrentPathIterator

ciD...	
ciDefaultParamToMatchFilter	ciDeleteUnreferencedObjects
ciDeleteModgenTopologies	ciDesignLCV
ciDeleteRuleGroup	
ciE...	
ciEnableAutoConstraintNotes	ciExpandName
ciExpandMembers	
ciF...	
ciFindOpenCellView	ciFindOpenCellViews
ciG...	
ciGetCellTermDefaultNetName	ciGetMembersOfType
ciGetCellView	ciGetObjectCellView
ciGetCellViewForObjectPath	ciGetOpenCellViews
ciGetConnectedInsts	ciGetRuleGroupByName
ciGetCustomFilterNames	ciGetRuleGroupName
ciGetDefaultNetName	ciGetRuleGroups
ciGetFoundryRules	ciGetWidgetProperties
ciGetMatchParam2DList	
ciH...	
ciHasCellAnyRegTerm	ciHierCompareConstraints
ciHaveSameBulkNets	ciHierUpdateConstraints
ciHierCompareConstraint	
ciI...	
ciIsNetSuperType	
ciL...	
ciLoadConfigXML	ciLoadIcons
ciLoadConfigXMLFromString	ciListEditors
ciLoadConstrFrom	ciListTypes
ciLoadDotCadenceFiles	ciListProcessRules

ciLoadIcon	ciLxComparisonReport
ciM...	
ciModgenMergeLayersFromArgs	ciModgenSplitFingers
ciModgenListFingerSplitCons	ciModgenTemplateFingerSplitPreDestroy
ciModgenRefreshStorage	
ciO...	
ciObjectIsInContext	ciOpenCellView
ciObjectListCon	ciOpenPanicCellView
ciObjectPathAndName	
ciP...	
ciPrintReport	ciPushConstraint
ciPullConstraint	ciPushConstraints
ciPullConstraints	
ciR...	
ciRefreshCellView	ciRemoveProcessRules
ciRegisterConstraintEditor	ciRemoveTrailingSlash
ciRegisterCustomDeviceFilter	ciReopenCellView
ciRegisterNetSuperType	ciReorderAssistants
ciRegTypeBindingParameter	ciResistorArrayUpdateRowColVal
ciRemoveConstrainedPinNetsFromRails	ciResolveBulkNet
ciRemoveHierarchicalNotes	ciRunMatchingConstraintsGenerator
ciRemoveLeadingSlash	
ciS...	
ciSelectedConstraints	ciSetModgenTopology
ciSelectedTemplates	ciSetSymmetricAxes
ciSetHaloOptions	ciSigTypeMatchesNetType
ciSetHaloPolicy	ciSimpleName
ciSetMaxHaloGroupSize	ciSortedOpenCellViews

ciT...	
ciTemplateAddCons	ciTemplateListParams
ciTemplateCreate	ciTemplatep
ciTemplateCreateDefinition	ciTemplateResetAllParams
ciTemplateCreateExpanded	ciTemplateResetParams
ciTemplateDefinitionExists	ciTemplateSetNote
ciTemplateDelete	ciTemplateSetStatus
ciTemplateDeleteCons	ciTemplateSortParamDefs
ciTemplateFind	ciTemplateUpdateParams
ciTemplateGetCache	ciToFloat
ciTemplateGetComment	ciTransferConstraintsInProgress
ciTemplateGetCreatedTime	ciTypeBindingParameter
ciTemplateGetDefName	ciTypeDefBaseType
ciTemplateGetName	ciTypeDefVersion
ciTemplateGetNote	ciTypeHasBindingParameter
ciTemplateGetStatus	ciTypeIsType
ciTemplateGetType	ciTypeIsUserDefined
ciTemplateListCon	ciTypeListCon
ciTemplateListParamNames	
ciU...	
ciUniqueMembers	ciUtilsAddNTimes
ciUnregisterAssistant	ciUtilsAddQuotes
ciUnregisterConstraintEditor	ciUtilsBuildString
ciUnregisterNetSuperType	ciUtilsMakeUnique
ciUnRegisterTerm	ciUtilsRemoveNils
ciUpdateHierarchicalNotes	ciUtilsRepeatNTimes
ciUprevEAConstrs	ciUtilsReplaceNils
ciW...	
ciWithinConstraint	

ciAddHierarchicalNotes

```
ciAddHierarchicalNotes(
     t_libName
     t_cellName
     t_viewName
     [ g_hierarchical ]
     )
     => t / nil
```

Description

Adds notes to the templates and constraints for a single cellview or all along the hierarchy beginning from the given cellview.

Note: In context of this SKILL function, notes are labels and annotations on the schematic.

See also ciUpdateHierarchicalNotes and ciRemoveHierarchicalNotes.

Arguments

t_libName	The library that contains the cellview to which notes need to be added.
t_cellName	The cell that contains the view to which notes need to be added.
t_viewName	The view to which notes need to be removed.
g_hierarchical	Determines whether the notes should be added to the whole hierarchy for the specified cellview. The valid values are `t` to add the notes all along the hierarchy and `nil` to add the notes only to the specified cellview.

Value Returned

`t`	Notes successfully added to the given cellview.
`nil`	Notes could not be added.

Examples

To add notes to the entire hierarchy starting from the specified cellview:

```
ciAddHierarchicalNotes("myLib" "myCellName" "myView" t)
```

ciAddLeadingSlash

```
ciAddLeadingSlash(
     t_name
     [ ?skipEmpty g_skipEmpty ]
     )
     => t_name
```

Description

Adds a forward slash (`/`) to the beginning of the passed string if it does not already have one.

Arguments

t_name	The name to be prefixed with a forward slash.
?skipEmpty *g_skipEmpty*	Determines whether empty strings should be skipped or not.

Value Returned

t_name	The name with a leading forward slash added.

Examples

```
ciAddLeadingSlash("NM1")
=> "/NM1"
ciAddLeadingSlash("/I1/NM1")
=> "/I1/NM1"
ciAddLeadingSlash("")
=> ""
ciAddLeadingSlash("" ?skipEmpty nil)
=> "/"
```

ciAddProcessRules

```
ciAddProcessRules(
     d_Id
     l_rules
     [ t_parameterType ]
     )
     => t / nil
```

Description

Adds the specified process rules to the front of the rules associated with the given object.

By default, the process rules for all objects are derived from the foundry constraint group in the technology database. These rules can however be overridden for specific objects or for a given design.

Process rule overrides on any object apply to other objects following a hierarchical precedence. For example, when applied to nets they will affect all routes and shapes that belong to that net, and when applied to routes they will affect all the shapes (paths, pathSegs) on that route. In Virtuoso, applications follow the precedence of: *wires* (paths/pathSegs/vias/guides) -> *routes* -> *nets* -> *net classes* -> *design*, when looking up constraints on objects (defaulting to foundry). The lookup is further guided by the ordering of constraints within any one `constraintGroup` (*hard* lookup) and the precedence ordering of the constraint groups themselves (*soft* lookup).

Note: The `ciAddProcessRules` SKILL command only applies the *hard* lookup find, that is only to those constraints that are within any single `constraintGroup`.

See also:

- The process rule overrides constraint in the *Virtuoso Unified Custom Constraints User Guide*.
- Process Rules for Objects in the *Virtuoso Unified Custom Constraints User Guide*.

Arguments

d_Id	A route or net object. **Note:** The `ciAddProcessRules` function also accepts cellviews.

l_rules

A list of elements:

`'('ruleName ruleLayers ruleValue)` or
`'('group 'groupSource groupName)`

- `ruleName` is a supported rule type

  Supported types are: `minWidth`, `minSpacing`, `validLayers`, `validVias`, `minNumCut`. For example:

  ```
  (type list(layers as strings)
  value_as_appropriate_type)
  ```

  - For `minWidth` type, spacing value should be a float distance.

  - For `minNumCut` type, the value should be an integer.

  - For `validVias` type, the value should be a list of def names as strings.

  - For `validLayers` type, the values should be a list of layer names as strings.

- `ruleLayer` is a string name of a layer or list of strings for multiple layers.

  The number of layers, on `list(layers as strings)`, will depend on the constraint. For `minSpacing`, `minNumCut` and `minWidth` allowed vias is 1 and layers will be 0.

- `groupSource` may be `'tech` or `'design`

  Also supported is a `group` keyword followed by where that `group` is stored, either `tech` or `design`, followed by the `group` name. For example:

  ```
  ciAddProcessRules(net list( list('group
  'tech "MyConstraintGroup")))
  ```

t_parameterType	Optional string argument. Depending on what the object type is, the valid values for this argument are: ■ Default - for nets and constraints that have a *Default* param (this is the default value) ■ InputTaper - for nets ■ OutputTaper - for nets ■ Reflexive - for constraints that have a *Within Group* parameter ■ TransReflexive - for constraints that have a *Group to Outside Group* parameter ■ Interchild - for nested net classes (constraints that have a *Group to Group* parameter) ■ Shielding - for shielding constraints (cons that have a *Shielding* parameter)

Value Returned

`t`	Process rules correctly added.
`nil`	Command failed.

Example

To add a spacing to *Group to Outside Group* on a netclass:

```
ciAddProcessRules(netClassCon list(list("minSpacing" "Metal1" .2 ))
"TransReflexive")
```

To define default values for the top and bottom layers of a constraint group called `myRG1`:

```
ciAddProcessRules(myRG1 '( ("minWidth" "Metal1" 1.0) ("minWidth" "Metal2" 1.2)
("validLayers" nil ("Metal1" "Metal2" "Metal3"))))
```

ciAddRuleGroup

```
ciAddRuleGroup(
     [ d_techID | d_cellViewID ]
     t_name
     )
     => t / nil
```

Description

Creates an `oaConstraint` group, with a given name, for technology or design files.

The first argument given should be either `d_techID` or `d_cellViewID`, followed by `t_name`.

Arguments

d_techID	The technology file identifier.
d_cellViewID	The cellview identifier.
t_name	The name to be applied to the `oaConstraintGroup`.

Value Returned

`t`	Successfully created the `oaConstraintGroup`.
`nil`	`oaConstraintGroup` not created.

Example

```
tf = techGetTechFile(geGetEditCellView ())
cg1 = ciAddRuleGroup(tf "cg1")
```

To create a new constraint group called `myRG1`:

```
cv=geGetEditCellView()
myRG1=ciAddRuleGroup(cv "myRG1")
```

The new constraint group can now be selected as the *Default Group* (*Constraint Group*) on the Constraint Parameter Editor. Default values for the `myRG1` parameters can be defined using the `ciAddProcessRules` command.

ciAddTrailingSlash

```
ciAddTrailingSlash(
     t_name
     [ ?skipEmpty g_skipEmpty ]
     )
     => t_name
```

Description

Adds a forward slash (`/`) to the end of the passed string if it does not already have one.

Arguments

t_name	The name to which the forward slash needs to be added at the end.
?skipEmpty *g_skipEmpty*	Determines whether empty strings should be skipped or not.

Value Returned

t_name	The name with a trailing forward slash added.

Examples

```
ciAddTrailingSlash("NM1")
=> "NM1/"
ciAddTrailingSlash("/I1/NM1/")
=> "/I1/NM1/"
ciAddTrailingSlash("")
=> ""
ciAddTrailingSlash("" ?skipEmpty nil)
=> "/"
```

ciAllCellViewsInHierarchy

```
ciAllCellViewsInHierarchy(
     d_cellViewDBId
     [ t_pathToCurrentCellView ]
     [ t_pathFilter ]
     [ t_depth ]
     [ s_predicate ]
     )
     => list / nil
```

Description

Returns a list of disembodied property lists containing the path and database ID of each cellview in the hierarchy, or to a specific depth starting from a given cellview and path to that cellview.

Arguments

d_cellViewDBId	The database ID of the current cellview.
t_pathToCurrentCellView	The path to the current cellview. This path will be prefixed to all hierarchical path strings.
t_pathFilter	The filter to show only the cellviews for a given path.
t_depth	The depth in the hierarchy up to which the search should be done. If this argument is specified, the result is retrieved up to that particular depth, otherwise all cellviews will be listed.
s_predicate	The function object or symbol that points to another function object. The function object should accept the following four arguments: `libName`, `cellName`, `viewName`, and `depth`. This `predicate` function is used to prune part of design when searching for all cellviews in the hierarchy.

Value Returned

list	List of all cellviews and paths below the current cellview up to the specified depth.
nil	Command failed.

Example

```
allCVs = ciAllCellViewsInHierarchy(geGetEditCellView())
((nil hierPath "" cellView db:0xf0f1e492)
 (nil hierPath "/I16" cellView db:0xf0f1b192)
 (nil hierPath "/I18" cellView db:0xf0f1b192)
 (nil hierPath "/I17" cellView db:0xf0f1b192)
 (nil hierPath "/I17/I22" cellView db:0xf0f1b194)
 (nil hierPath "/I17/I23" cellView db:0xf0f1b198)
 (nil hierPath "/I15" cellView db:0xf0f1ac12)
)

I17cv = nthelem(4 allCVs)

I17CVs = ciAllCellViewsInHierarchy(I17cv "/I17")
((nil hierPath "/I17" cellView db:0xf0f1b192)
 (nil hierPath "/I17/I22" cellView db:0xf0f1b194)
 (nil hierPath "/I17/I23" cellView db:0xf0f1b198)
)

allCVsI17 = ciAllCellViewsInHierarchy(geGetEditCellView() "" "/I17")
((nil hierPath "" cellView db:0xf0f1e492)
 (nil hierPath "/I17" cellView db:0xf0f1b192)
 (nil hierPath "/I17/I22" cellView db:0xf0f1b194)
 (nil hierPath "/I17/I23" cellView db:0xf0f1b198)
)

allCVsDepth1 = ciAllCellViewsInHierarchy(geGetEditCellView() "" "" 1)
((nil hierPath "" cellView db:0xf0f1e492)
 (nil hierPath "/I16" cellView db:0xf0f1b192)
 (nil hierPath "/I18" cellView db:0xf0f1b192)
 (nil hierPath "/I17" cellView db:0xf0f1b192)
 (nil hierPath "/I15" cellView db:0xf0f1ac12)
)
```

```
;;Only look into cellview which are part of the library AnalogLib.
procedure(myPredicate(libName cellName viewName depth)
    !exists(x '(("ether_adcflash_RAD90" "adc_sample_hold" "schematic")
        ("ether_adcflash_RAD90" "adcflash_comparator_actr" "schematic")
        ("ether_adcflash_RAD90" "and2_1x_hv" "schematic")
        ("ether_adcflash_RAD90" "clkbuf_2x_hv" "schematic")
        ("ether_adcflash_RAD90" "inv_2x_hv" "schematic")
        ("ether_adcflash_RAD90" "nand2_2x_hv" "schematic")
        ("ether_adcflash_RAD90" "or2_2x_hv" "schematic")
        ("ether_adcflash_RAD90" "sheet_a" "symbol")
        ("ether_adcflash_RAD90" "sheet_aa" "symbol")
        ("ether_adcflash_RAD90" "sheet_b" "symbol"))
    equal(x list(l c v)))
)

ciAllCellViewsInHierarchy(cv currentHierPath pathFilter depth 'myPredicate)
ciAllCellViewsInHierarchy(geGetEditCellView(window(3)) "" "" -1 'testPredicate)
((nil hierPath "" cellView db:0x17f0b69a)
    (nil hierPath "/ADCFLASH_REF_LADDER" cellView db:0x17f0711a)
    (nil hierPath "/ADCFLASH_REF_LADDER/I2" cellView db:0x17f0af1a)
    (nil hierPath "/ADCFLASH_REF_LADDER/R1" cellView db:0x17f0949a)
    (nil hierPath "/ADCFLASH_REF_LADDER/R11" cellView db:0x17f0949a)
    (nil hierPath "/ADCFLASH_REF_LADDER/R12<4:0>" cellView db:0x17f09a9a)
    (nil hierPath "/ADCFLASH_REF_LADDER/R13<1:0>" cellView db:0x17f09a9a)
    (nil hierPath "/ADCFLASH_REF_LADDER/R17<1:0>" cellView db:0x17f09a9a)
    (nil hierPath "/ADCFLASH_REF_LADDER/R18<4:0>" cellView db:0x17f09a9a)
    (nil hierPath "/ADCFLASH_REF_LADDER/R19<4:0>" cellView db:0x17f09a9a)
    (nil hierPath "/ADCFLASH_REF_LADDER/R20<1:0>" cellView db:0x17f09a9a)
    (nil hierPath "/ADCFLASH_REF_LADDER/R21<1:0>" cellView db:0x17f09a9a)
    (nil hierPath "/ADCFLASH_REF_LADDER/R3<1:0>" cellView db:0x17f09a9a)
    (nil hierPath "/ADCFLASH_REF_LADDER/R7<1:0>" cellView db:0x17f09a9a)
    (nil hierPath "/ADCFLASH_REF_LADDER/R8<4:0>" cellView db:0x17f09a9a)
)

ciAllCellViewsInHierarchy(cv currentHierPath pathFilter depth)
ciAllCellViewsInHierarchy(geGetEditCellView(window(3)) "" "" -1 )
((nil hierPath "" cellView db:0x17f0b69a)
    (nil hierPath "/ADCFLASH_REF_LADDER" cellView db:0x17f0711a)
    (nil hierPath "/ADCFLASH_SAMPLE_HOLD" cellView db:0x17f06a1a)
    (nil hierPath "/I22" cellView db:0x17f0619a)
```

```
(nil hierPath "/I37" cellView db:0x17f0531a)
(nil hierPath "/I38" cellView db:0x17f04d1a)
(nil hierPath "/I39" cellView db:0x17f04d1a)
(nil hierPath "/I4" cellView db:0x17f0619a)
(nil hierPath "/I40" cellView db:0x17f0499a)
(nil hierPath "/I41" cellView db:0x17f0499a)
(nil hierPath "/I42" cellView db:0x17f0531a)
(nil hierPath "/I5" cellView db:0x17f0619a)
(nil hierPath "/I6" cellView db:0x17f0619a)
(nil hierPath "/I7" cellView db:0x17f0619a)
(nil hierPath "/I8" cellView db:0x17f0619a)
(nil hierPath "/I9" cellView db:0x17f0619a)
(nil hierPath "/ADCFLASH_REF_LADDER/I2" cellView db:0x17f0af1a)
(nil hierPath "/ADCFLASH_REF_LADDER/R1" cellView db:0x17f0949a)
(nil hierPath "/ADCFLASH_REF_LADDER/R11" cellView db:0x17f0949a)
(nil hierPath "/ADCFLASH_REF_LADDER/R12<4:0>" cellView db:0x17f09a9a)
(nil hierPath "/ADCFLASH_REF_LADDER/R13<1:0>" cellView db:0x17f09a9a)
(nil hierPath "/ADCFLASH_REF_LADDER/R17<1:0>" cellView db:0x17f09a9a)
(nil hierPath "/ADCFLASH_REF_LADDER/R18<4:0>" cellView db:0x17f09a9a)
(nil hierPath "/ADCFLASH_REF_LADDER/R19<4:0>" cellView db:0x17f09a9a)
(nil hierPath "/ADCFLASH_REF_LADDER/R20<1:0>" cellView db:0x17f09a9a)
(nil hierPath "/ADCFLASH_REF_LADDER/R21<1:0>" cellView db:0x17f09a9a)
(nil hierPath "/ADCFLASH_REF_LADDER/R3<1:0>" cellView db:0x17f09a9a)
(nil hierPath "/ADCFLASH_REF_LADDER/R7<1:0>" cellView db:0x17f09a9a)
(nil hierPath "/ADCFLASH_REF_LADDER/R8<4:0>" cellView db:0x17f09a9a)
(nil hierPath "/ADCFLASH_SAMPLE_HOLD/I58" cellView db:0x17f04d1a)
(nil hierPath "/I22/I12" cellView db:0x17f0431a)
(nil hierPath "/I22/I13" cellView db:0x17f0431a)
(nil hierPath "/I22/I16" cellView db:0x17f03d9a)
(nil hierPath "/I22/I17" cellView db:0x17f03d9a)
(nil hierPath "/I22/R2" cellView db:0x17f09a9a)
(nil hierPath "/I22/R3" cellView db:0x17f09a9a)
(nil hierPath "/I4/I12" cellView db:0x17f0431a)
(nil hierPath "/I4/I13" cellView db:0x17f0431a)
(nil hierPath "/I4/I16" cellView db:0x17f03d9a)
(nil hierPath "/I4/I17" cellView db:0x17f03d9a)
(nil hierPath "/I4/R2" cellView db:0x17f09a9a)
(nil hierPath "/I4/R3" cellView db:0x17f09a9a)
(nil hierPath "/I5/I12" cellView db:0x17f0431a)
(nil hierPath "/I5/I13" cellView db:0x17f0431a)
(nil hierPath "/I5/I16" cellView db:0x17f03d9a)
```

```
    (nil hierPath "/I5/I17" cellView db:0x17f03d9a)
    (nil hierPath "/I5/R2" cellView db:0x17f09a9a)
    (nil hierPath "/I5/R3" cellView db:0x17f09a9a)
    (nil hierPath "/I6/I12" cellView db:0x17f0431a)
    (nil hierPath "/I6/I13" cellView db:0x17f0431a)
    (nil hierPath "/I6/I16" cellView db:0x17f03d9a)
    (nil hierPath "/I6/I17" cellView db:0x17f03d9a)
    (nil hierPath "/I6/R2" cellView db:0x17f09a9a)
    (nil hierPath "/I6/R3" cellView db:0x17f09a9a)
    (nil hierPath "/I7/I12" cellView db:0x17f0431a)
    (nil hierPath "/I7/I13" cellView db:0x17f0431a)
    (nil hierPath "/I7/I16" cellView db:0x17f03d9a)
    (nil hierPath "/I7/I17" cellView db:0x17f03d9a)
    (nil hierPath "/I7/R2" cellView db:0x17f09a9a)
    (nil hierPath "/I7/R3" cellView db:0x17f09a9a)
    (nil hierPath "/I8/I12" cellView db:0x17f0431a)
    (nil hierPath "/I8/I13" cellView db:0x17f0431a)
    (nil hierPath "/I8/I16" cellView db:0x17f03d9a)
    (nil hierPath "/I8/I17" cellView db:0x17f03d9a)
    (nil hierPath "/I8/R2" cellView db:0x17f09a9a)
    (nil hierPath "/I8/R3" cellView db:0x17f09a9a)
    (nil hierPath "/I9/I12" cellView db:0x17f0431a)
    (nil hierPath "/I9/I13" cellView db:0x17f0431a)
    (nil hierPath "/I9/I16" cellView db:0x17f03d9a)
    (nil hierPath "/I9/I17" cellView db:0x17f03d9a)
    (nil hierPath "/I9/R2" cellView db:0x17f09a9a)
    (nil hierPath "/I9/R3" cellView db:0x17f09a9a)
)
```

ciAxisCreate

```
ciAxisCreate(
     u_cache
     t_axisName
     [ l_parameterList ]
     )
     => t / nil
```

Description

Creates a new axis.

Arguments

u_cache	The constraint cache where the axis is located.
t_axisName	The axis name.
l_parameterList	The parameter_list for the axis (optional argument).

Allowed parameters are:

```
direction = "horizontal" | "vertical"
axisLocation = "any" | "fixed"
coordinate (only valid if location is
"fixed")

Examples:

; Create a movable horizontal axis.
ciAxisCreate(cache
              "hAxis"
              '(("direction" "horizontal")
("axisLocation" "any")))

; Create a fixed, vertical axis.
ciAxisCreate(cache
              "vAxis"
              '(("direction" "vertical")
                ("axisLocation" "fixed")
                ("coordinate" 10)))
```

Value Returned

`t`	Successfully created an axis.
`nil`	Axis not created.

ciAxisDelete

```
ciAxisDelete(
     u_cache
     t_axisName
     [ g_forceDelete ]
     )
     => t / nil
```

Description

Deletes an axis from the specified cache.

Arguments

u_cache	The constraint cache where the axis is located.
t_axisName	The axis name.
g_forceDelete	By default, an axis will not be deleted if it is being used by one or more constraints. Pass t to force the axis to be destroyed. Any constraints attached to the axis will be updated to use a default axis.

Value Returned

t	Successfully deletes an axis.
nil	Axis not deleted.

ciAxisExists

```
ciAxisExists(
     u_cache
     t_axisName
     )
     => t / nil
```

Description

Returns `t` (true) if the named axis exists in the cache. Access to axes is only provided through the axis name. No direct reference to an axis object is provided.

Arguments

u_cache	The constraint cache where the axis is located.
t_axisName	The axis name.

Value Returned

t	The named axis exists.
nil	The named axis does not exist.

ciAxisListCon

```
ciAxisListCon(
     u_cache
     t_axisName
     )
     => u_constraint / nil
```

Description

Lists all the constraints that are attached to a given axis.

Arguments

u_cache	The constraint cache where the axis is located.
t_axisName	The axis name.

Value Returned

u_constraint	A list of all the constraint types found in the cache.
nil	No axes exist in the cache or no constraint is associated with the given axis.

ciAxisListParams

```
ciAxisListParams(
     u_cache
     t_axisName
     [ g_includeDefaults ]
     )
     => parameterList / nil
```

Description

Lists the parameters associated with an axis.

Arguments

u_cache	The constraint *cache* where the axis is located.
t_axisName	The axis name.
g_includeDefaults	Optional argument. A Boolean value (`t` if unspecified) that includes or excludes parameters with default values when `t` or `nil` respectively.

Value Returned

parameter_list	List of constraint parameters.
`nil`	No constraint parameters found for axis.

ciAxisReplaceParams

```
ciAxisReplaceParams(
     u_cache
     t_axisName
     l_parameterList
     )
     => t / nil
```

Description

Replaces all axis constraint parameters with a new set of parameters.

Arguments

u_cache	The constraint *cache* where the axis is located.
t_axisName	The axis name.
l_parameterList	The parameter_list for the axis.

Allowed parameters are:

```
direction = "horizontal" | "vertical"
axisLocation = "any" | "fixed"
coordinate (only valid if location is
"fixed")

Examples:

; Replace a movable horizontal axis.
ciAxisReplaceParams(cache
              "hAxis"
              '(("direction" "horizontal")
("axisLocation" "any")))

; Replace a fixed, vertical axis.
ciAxisReplaceParams(cache
              "vAxis"
              '(("direction" "vertical")
                ("axisLocation" "fixed")
                ("coordinate" 10)))
```

Value Returned

`t`	The parameters have been successfully replaced.
`nil`	The axis does not exist or the parameters are illegal.

ciCacheCallbackRegister

```
ciCacheCallbackRegister(
     l_callbackPropList
     )
     => t / nil
```

Description

Registers a callback.

Arguments

l_callbackPropList

A disembodied property list containing the callback properties in the following format:

```
'(nil
callback callbackName
type ciPreSaveCallback|ciPostSaveCallback
enable t|nil)
```

Here,

- `callbackName` is the name of the callback function that has been defined before the registration occurs. If this is not the case, the function returns `nil` and a warning message is displayed.
- `type` can have one of the following values: `ciPreSaveCallback` or `ciPostSaveCallback`. If you specify a different value, the function returns `nil` and a warning message is displayed.
- `enable` can be `t` or `nil`.
    - If it is set to `t`, the callback is enabled; otherwise, it is disabled.
    - If `enable` is omitted, the callback is registered, but disabled.

- ❑ If `enable` is set to a value other than `t` or `nil`, it is reset to `nil` and a warning message is displayed.
- Additional properties added to the list are ignored.

Value Returned

`t`	The callback was registered successfully.
`nil`	The registration failed and the callback registry was not updated.

Examples

```
;; Definition of a pre-save callback:
    (defun myPreSaveCallback (cache)
        ;; Do something useful with the cache here.
        ;; Do not call any save function here.
        (printf "Cache for design %L is about to be saved." cache->design))

;;Definition of a post-save callback:
    (defun myPostSaveCallback (cache saved)
        ;; Do something useful with the saved cache.
        ;; Do not call any save function here.
        ;; For the time being 'saved' always retuns 't'
        (printf "Cache for design %L has been saved [%L].\n" cache->design saved))

;; Definition of the callback lists
    preSaveCB  = '(nil callback myPreSaveCallback  type ciPreSaveCallback  enable
    t)
    postSaveCB = '(nil callback myPostSaveCallback type ciPostSaveCallback)

;; Register the callbacks
    (mapcar 'ciCacheCallbackRegister (list preSaveCB postSaveCB))
    ;; If the callbacks were not registered already, this call returns (t t)
    ;; In this case, a second call will return (nil nil) because the callback
    ;; registry will not be changed.

    ;; Currently only the preSaveCB will be called on saved, because the postSaveCB
    is disabled
```

```
    ;; To enable it, we need to call ciCacheCallbackUpdate.
        postSaveCB->enable = t
        (ciCacheCallbackUpdate postSaveCB)
        ;; returns 't'. Now the callback is enabled. A second call returns 'nil'
        because the callback registry will not be changed.

;; To unregister the callbacks
    (mapcar 'ciCacheCallbackUnregister (list preSaveCB postSaveCB))
    ;; returns (t t). A second call returns (nil nil), because the callback registy
    ;; will not be changed.
    ;; Calls to ciCacheCallbackUpdate for these callbacks will return nil.
    ;; These callbacks need to be registered again to be re-enabled.
```

ciCacheCallbackUnregister

```
ciCacheCallbackUnregister(
     l_callbackPropList
     )
     => t / nil
```

Description

Unregisters a previously registered callback.

Arguments

l_callbackPropList

A disembodied property list containing the callback properties in the following format:

```
'(nil
callback callbackName
type ciPreSaveCallback|ciPostSaveCallback
enable t|nil)
```

Here,

- *callbackName* is the name of the callback function that has been defined before the registration occurs. If this is not the case, the function returns `nil` and a warning message is displayed.
- `type` can have one of the following values: `ciPreSaveCallback` or `ciPostSaveCallback`. If you specify a different value, the function returns `nil` and a warning message is displayed.
- `enable` can be `t` or `nil`.
  - If it is set to `t`, the callback is enabled; otherwise, it is disabled.
  - If `enable` is omitted, the callback is registered, but disabled.

- If `enable` is set to a value other than `t` or `nil`, it is reset to `nil` and a warning message is displayed.

- Additional properties added to the list are ignored.

Value Returned

`t`	The callback was unregistered successfully.
`nil`	The operation failed and the callback registry was not updated.

Example

Refer to the Examples given in the ciCacheCallbackRegister section.

ciCacheCallbackUpdate

```
ciCacheCallbackUpdate(
     l_callbackPropList
     )
     => t / nil
```

Description

Updates a registered callback.

Arguments

l_callbackPropList

A disembodied property list containing the callback properties in the following format:

```
'(nil
callback callbackName
type ciPreSaveCallback|ciPostSaveCallback
enable t|nil)
```

Here,

- *callbackName* is the name of the callback function that has been defined before the registration occurs. If this is not the case, the function returns `nil` and a warning message is displayed.
- `type` can have one of the following values: `ciPreSaveCallback` or `ciPostSaveCallback`. If you specify a different value, the function returns `nil` and a warning message is displayed.
- `enable` can be `t` or `nil`.
    - If it is set to `t`, the callback is enabled; otherwise, it is disabled.
    - If `enable` is omitted, the callback is registered, but disabled.

- If `enable` is set to a value other than `t` or `nil`, it is reset to `nil` and a warning message is displayed.

- Additional properties added to the list are ignored.

Value Returned

`t`	The registered callback was updated.
`nil`	The operation failed and the callback registry was not updated.

Example

Refer to the Examples given in the ciCacheCallbackRegister section.

ciCacheCellName

```
ciCacheCellName(
     u_cache
     )
     => t_cellName
```

Description

Returns the cell name of the specified constraint cache. The cache library, cell, and view refers to the schematic, configuration, physical configuration, or layout view that the constraint cache is associated with.

Arguments

u_cache	The constraint cache ID.

Value Returned

t_cellName	The cell name of the associated schematic, configuration, physical configuration, or layout.

Example

Return the cell name of the current constraint cache:

```
ciCacheCellName(ciGetCellView())
=> "block1"
```

ciCacheConstraintCellName

```
ciCacheConstraintCellName(
     u_cache
     )
     => t_cellName
```

Description

Returns the constraint cell name of the specified constraint cache. The constraint library, cell, and view refers to the storage location of the constraints, which might be a constraint view or layout.

Arguments

u_cache	The constraint cache ID.

Value Returned

t_cellName	The cell name of the constraint view or layout.

Example

Return the constraint cell name of the current constraint cache:

```
ciCacheConstraintCellName(ciGetCellView())
=> "block1"
```

ciCacheConstraintLibName

```
ciCacheConstraintLibName(
     u_cache
     )
     => t_libName
```

Description

Returns the constraint library name of the specified constraint cache. The constraint library, cell, and view refers to the storage location of the constraints, which might be a constraint view or layout.

Arguments

u_cache	The constraint cache ID.

Value Returned

t_libName	The library name of the constraint view or layout.

Example

Return the constraint library name of the current constraint cache:

```
ciCacheConstraintLibName(ciGetCellView())
=> "overview"
```

ciCacheConstraintViewName

```
ciCacheConstraintViewName(
     u_cache
     )
     => t_viewName
```

Description

Returns the constraint view name of the specified constraint cache. The constraint library, cell, and view refers to the storage location of the constraints, which might be a constraint view or layout.

Arguments

u_cache	The constraint cache ID.

Value Returned

t_viewName	The view name of the constraint view or layout.

Example

Return the constraint view name of the current constraint cache:

```
ciCacheConstraintViewName(ciGetCellView())
=> "constrBlock"
```

ciCacheDiscardEdits

```
ciCacheDiscardEdits(
     u_cache
     )
     => u_cache / nil
```

Description

Discards changes made to the constraint cache in the memory and returns a new constraint cache ID. This function does not apply to layout constraint caches.

Arguments

u_cache	The constraint cache ID.

Value Returned

u_cache	A new constraint cache ID if the changes made to the constraint cache could be discarded successfully.
nil	The discard changes operation failed.

Example

Discard the memory changes made to the current constraint cache:

```
cache = ciGetCellView()
  when( ciCacheIsModified(cache)
    cache = ciCacheDiscardEdits(cache)
  )
```

ciCacheFind

```
ciCacheFind(
     d_dbCellViewId | t_libName t_cellName t_viewName
     )
     => u_cache / nil
```

Description

Finds an existing constraint cache for a given design specified using library, cell, and view names. Alternatively, pass the `dbCellViewId` of the required design.

Arguments

d_dbCellViewId	The cellview ID of the design.
t_libName	The library to locate constraint cache in.
t_cellName	The cell to locate constraint cache in.
t_viewName	The view to locate constraint cache in.

Value Returned

u_cache	Returns the constraint cache if the cache has already been built (see `u_cache`).
`nil`	No cache found.

Examples

Find a cache using lib/cell/view names:

```
cache = ciCacheFind("myLib" "myDesign" "schematic")
```

Find a cache using the cellview in the current editor window:

```
cache = ciCacheFind(geGetEditCellView())
```

ciCacheGet

```
ciCacheGet(
     d_dbCellViewId | t_libName t_cellName t_viewName
     )
     => u_cache / nil
```

Description

Finds an existing constraint `cache`. If it cannot find an existing cache, it creates and populates a cache for the given design, using library, cell, and view names. Alternatively, pass the `dbCellViewId` of the required design.

Arguments

d_dbCellViewId	The cellview ID of the design.
t_libName	The library to locate or create constraint cache in.
t_cellName	The cell to locate or create constraint cache in.
t_viewName	The view to locate or create constraint cache in.

Value Returned

u_cache	Returns the constraint cache if the cache has already been built through a previous call to `ciCacheGet`, otherwise builds the cache for the given cell view and returns it (see `u_cache`).
nil	If the specified `library`, `cell`, or `view` does not exist or cannot be associated with a constraint cache (that is, not a layout, schematic, config or physConfig), then the associated constraint view cannot be opened or created, such as if the cell is read-only and the constraint view does not exist.

Example

```
cv = geGetEditCellView()

ciCacheGet(cv)
ci:0x252c2d90
```

```
ciCacheGet(cv->libName cv->cellName cv->viewName)
ci:0x252c2d90
```

ciCacheGetAllNetNames

```
ciCacheGetAllNetNames(
     g_cache
     [ ?expandNames g_expandNames ]
     )
     => l_result
```

Description

Returns a sorted list of all the net names in the cellview associated with the constraints cache. By default, the iterated net names are expanded.

Arguments

g_cache	The constraints cache.
?expandNames *g_expandNames*	Used to control whether the iterated names should be expanded. By default, it is set to `t`.

Value Returned

l_result	Returns a sorted list of the net names in the cellview. The expansion of iterated net names depends on the Boolean value set for the *g_expandNames* argument.

Examples

- To get a list of all net names in the cellview and expand the iterated net names:

```
cache = ciGetCellView()
ciCacheGetAllNetNames(cache)
( "netA" "netB" "netC<0>" "netC<1>" "netD" )
```

- To get a list of all net names in the cellview and ***not*** expanded iterated net names:

```
ciCacheGetAllNetNames(cache ?expandNames nil)
( "netA" "netB" "netC<0:1>" "netD" )
```

ciCacheGetCellView

```
ciCacheGetCellView(
     g_cache
     )
     => d_cellviewID / nil
```

Description

Returns the cellview associated to the specified cache.

Arguments

g_cache	The constraints cache.

Value Returned

d_cellviewID	Returns cellview ID.
nil	No cellview was found.

Examples

```
cv = ciCacheGetCellView(ciGetCellView())
dbFindAnyInstByName(cv "instanceName")
```

ciCacheGetEnabledNotifications

```
ciCacheGetEnabledNotifications(
    )
    => t_listNotifications / nil
```

Description

Returns a list of cache notifications that ciCacheNotifications enabled.

Arguments

None

Value Returned

t_listNotifications	List of notification symbols is returned if notifications have been enabled.
`nil`	None of the notifications have been enabled.

Examples

```
ciCacheNotifications('disable '(NotifyAll))
t
ciCacheGetEnabledNotifications()
nil
```

Returns `nil` because all notifications are disabled.

```
ciCacheNotifications('enable '(WarnDesignInfoUpdates InfoOnCacheEditModeChanges))
t
ciCacheGetEnabledNotifications()
(InfoOnCacheEditModeChanges WarnDesignInfoUpdates)
```

Returns a list of notification symbols that are enabled.

ciCacheIsLayout

```
ciCacheIsLayout(
     g_cache
     )
     => t / nil
```

Description

Returns `t` if the passed cache is a layout cache.

Arguments

g_cache	The constraint cache.

Value Returned

`t`	If the passed cache is a layout cache.
`nil`	If the passed cache is not a layout cache.

Example

```
schCache = ciCacheGet("amsPLL" "vco" "schematic")
         layCache = ciCacheGet("amsPLL" "vco" "layout")

         ciCacheIsLayout(schCache)
         nil
```

Retuns `nil` as the passed cache is not a layout cache.

```
         ciCacheIsLayout(layCache)
         t
```

Returns `t` as the passed cache is a layout cache.

ciCacheIsModified

```
ciCacheIsModified(
     u_cache
     )
     => t / nil
```

Description

Checks whether a constraint cache has been modified.

Arguments

u_cache	The constraint cache ID.

Value Returned

`t`	The constraint cache is modified.
`nil`	The constraint cache is not modified.

Example

The following saves a constraint cache if it has been modified:

```
cache = ciGetCellView()
  when( ciCacheIsModified(cache)
    ciCacheSave(cache)
  )
```

ciCacheIsWritable

```
ciCacheIsWritable(
     u_cache
     )
     => t / nil
```

Description

Checks whether a constraint cache is writable.

Arguments

u_cache	The constraint cache ID.

Value Returned

`t`	The constraint cache is writable.
`nil`	The constraint cache is read only.

Example

```
cache = ciGetCellView()
  unless( ciCacheIsWritable(cache)
    cache = ciCacheMakeEditable(cache)
  )
```

Makes a constraint cache editable if it is read only.

ciCacheLCV

```
ciCacheLCV(
     u_cache
     )
     => l_libCellViewNames
```

Description

Returns a list containing the library, cell, and view names of the specified constraint cache. The cache library, cell, and view refers to the schematic, configuration, physical configuration, or layout view that the constraint cache is associated with.

Arguments

u_cache	The constraint cache ID.

Value Returned

l_libCellViewNames	List comprising the library, cell and view names of the associated schematic, configuration, physical configuration or layout returned in the following format:
	`( t_libName t_cellName t_viewName )`
	For example:
	`( "overview" "block1" "schematic" )`

Example

```
ciCacheLCV(ciGetCellView())
=> ("overview" "block1" "physConfig")
```

Gets the library, cell, and view names of the current constraint cache.

ciCacheLibName

```
ciCacheLibName(
     u_cache
     )
     => t_libName
```

Description

Returns the library name of the specified constraint cache. The cache library, cell, and view refers to the schematic, configuration, physical configuration, or layout view that the constraint cache is associated with.

Arguments

u_cache	The constraint cache ID.

Value Returned

t_libName	The library name of the associated schematic, configuration, physical configuration, or layout.

Example

Return the library name of the current constraint cache:

```
ciCacheLibName(ciGetCellView())
=> "overview"
```

ciCacheListAxesNames

```
ciCacheListAxesNames(
     u_cache
     )
     => t_axesNames / nil
```

Description

Lists all axes names in a cache. Access to axes is only provided through the axis name, no direct reference to an axis object is provided.

Arguments

u_cache	The constraint cache where the axes are enumerated. See also `cache`.

Value Returned

t_axesNames	Returns a list of axes name strings.
`nil`	No axes exist in the cache.

ciCacheListCon

```
ciCacheListCon(
     u_cache
     [ g_includeOutOfContext ]
     )
     => l_constraintList / nil
```

Description

Lists all the constraints from a given cache.

Arguments

u_cache	See `cache`.
g_includeOutOfContext	Set to include out of context constraints (constraints not in the current view).

Value Returned

l_constraintList	List of all constraints in given cache.
`nil`	No constraints found.

ciCacheListConstrainedObjects

```
ciCacheListConstrainedObjects(
     u_cache
     [ g_includeOutOfContext ]
     )
     => l_designObjectList / nil
```

Description

Lists the name and type of all constrained objects.

Arguments

u_cache	See `cache`.
g_includeOutOfContext	If this optional argument is specified and set to `t`, the out of context constraints will be listed.

Value Returned

l_designObjectList	A list of all objects that are constrained. When the optional `parameter` `includeOutOfContext` is set to `t`, the out of context object will also be listed.
`nil`	No objects found.

ciCacheListConstrainedObjectNames

```
ciCacheListConstrainedObjectNames(
     u_cache
     [ g_includeOutOfContext ]
     )
     => l_objectNameList / nil
```

Description

Lists the names of the constrained objects (without the type of object).

Arguments

u_cache	See `cache`.
g_includeOutOfContext	If this optional argument is specified and set to `t`, out of context constraints will be listed.

Value Returned

l_objectNameList	A list of strings with all names of objects that are constrained. When the optional `parameter includeOutOfContext` is set to `t`, the out of context object will also be listed.
`nil`	No objects found.

ciCacheListTemplates

```
ciCacheListTemplates(
     u_cache
     [ t_templateType ]
     )
     => l_templateIdList / nil
```

Description

Lists all templates, of an optional given type, from a cache.

Arguments

u_cache	See `cache`.
t_templateType	Specifies a given template type.

Value Returned

l_templateIdList	A list of all templates found in the cache.
`nil`	No templates found.

ciCacheListTypeNames

```
ciCacheListTypeNames(
     u_cache
     )
     => l_constraintTypeList / nil
```

Description

Lists all the names of the constraint types that are legal for the given cache.

Arguments

u_cache	See `cache`.

Value Returned

l_constraintTypeList	A list of all constraint type names. The complete constraint type set listed in `constraint_type` that are also legal constraints for the current cache type.
`nil`	No constraint types found.

ciCacheListTypes

```
ciCacheListTypes(
     u_cache
     )
     => l_constraintTypeList / nil
```

Description

Lists all constraint types in a cache.

Arguments

u_cache	See `cache`.

Value Returned

l_constraintTypeList	A list of all the constraint types found in the cache.
`nil`	No constraint types found.

ciCacheMakeEditable

```
ciCacheMakeEditable(
     u_cache
     )
     => u_cache / nil
```

Description

Reopens the constraint cache for edit and returns a new constraint cache ID. This function does not apply to layout constraint caches.

Arguments

u_cache	The constraint cache ID.

Value Returned

u_cache	The new constraint cache ID if the constraint cache could be reopened in edit mode.
`nil`	The constraint cache could not be reopened in edit mode.

Example

```
cache = ciGetCellView()
  when( null(ciCacheIsWritable(cache))
    cache = ciCacheMakeEditable(cache)
  )
```

Reopens the current constraint cache for editing.

ciCacheMakeReadOnly

```
ciCacheMakeReadOnly(
     u_cache
     )
     => u_cache / nil
```

Description

Reopens the constraint cache in read-only mode and returns a new constraint cache ID. Any in modifications to the memory are discarded. This function does not apply to layout constraint caches.

Arguments

u_cache	The constraint cache ID.

Value Returned

u_cache	The new constraint cache ID if the constraint cache could be reopened in read-only mode.
nil	The constraint cache could not be reopened in read-only mode.

Example

```
cache = ciGetCellView()
  when( ciCacheIsWritable(cache)
    cache = ciCacheMakeReadOnly(cache)
  )
```

Reopens the current constraint cache in read mode.

ciCacheNeedRefresh

```
ciCacheNeedRefresh(
     u_cache
     )
     => t / nil
```

Description

Checks if the constraint cache has changed on the disk and needs to be refreshed.

Arguments

u_cache	The constraint cache ID.

Value Returned

`t`	The constraint cache needs to be refreshed from the disk.
`nil`	The constraint cache does not need to be refreshed.

Example

The following reports if the current constraint cache needs to be refreshed from the disk:

```
cache = ciGetCellView()
  when( ciCacheNeedRefresh(cache)
    info("Cache %L constraint view %L need refresh\n" ciCacheLCV(cache)
ciConstraintLCV(cache))
  )
Cache ("overview" "block1" "physConfig") constraint view ("overview" "block1"
"constrBlock") need refresh
```

The following reports how many contraint caches need to be refreshed from the disk:

```
  needrefresh = setof(c ciGetOpenCellViews() ciCacheNeedRefresh(c))
  when( needrefresh
    info("%d constraint caches need refresh\n" length(needrefresh))
  )
  2 constraint caches need refresh
```

ciCacheNotifications

```
ciCacheNotifications(
     s_enable
     l_notifications
     )
     => t / nil
```

Description

Enables or disables notifications related to the constraint cache. This function helps CAD teams in debugging.

Arguments

s_enable	Sets the function in enable or disable mode. The valid values are `'enable` and `'disable`.
l_notifications	Lists the symbols for selecting the notifications that need to be enabled or disabled. The valid notification symbols are as following: ■ `InfoOnCacheEditModeChanges` Notifies the users whenever the cache is switched from read to edit mode and conversely. ■ `WarnDesignInfoUpdates` Gives a warning whenever the schematic cache design information is updated except for config views. ■ `WarnDesignInfoConfigUpdates` Gives a warning whenever the schematic design information of a config view is updated. ■ `NotifyAll` Enables or disables all notifications.

Value Returned

`t`	The notifications were enabled or disabled successfully.
`nil`	The notifications were already enabled or disabled. **Note:** An error message is displayed if the argument values are incorrect.

Examples

```
ciCacheNotifications('enable '(WarnDesignInfoUpdates InfoOnCacheEditModeChanges))
t
```

Enables notifications for the edit and read mode switches in cache and any updates to the schematic cache design information excluding the config views.

```
ciCacheNotifications('enable '(WarnDesignInfoUpdates ))
nil
```

Returns `nil` because `WarnDesignInfoUpdates` is already enabled.

```
ciCacheNotifications('disable '(WarnDesignInfoUpdates ))
t
```

Returns `t` after disabling `WarnDesignInfoUpdates` notifications.

```
ciCacheNotifications('disable '(WarnDesignInfoUpdates ))
nil
```

Returns `nil` because `WarnDesignInfoUpdates` notifications are already disabled.

```
ciCacheNotifications('enable '(NotifyAll))
t
```

Enables all notifications and returns `t`.

```
ciCacheNotifications('disable '(NotifyAll))
t
```

Disables all notifications and returns `t`.

ciCachep

```
ciCachep(
     g_value
     )
     => t / nil
```

Description

Checks if an object is a valid constraint cache ID.

Arguments

g_value	Specifies the object to be checked.

Value Returned

`t`	*g_value* is a valid constraint cache ID.
`nil`	*g_value* is not a valid constraint cache ID.

Example

```
ciCachep(
       obj
       )
=> t
```

As shown in the example above, the object, `obj` is a valid constraint cache.

ciCachePurge

```
ciCachePurge(
     u_cache
     )
     => t / nil
```

Description

Purges a constraint cellview from virtual memory.

Arguments

u_cache	Specifies the constraint cacheID. See `cache`.

Value Returned

`t`	Constraint cellview successfully purged.
`nil`	Command failed.

ciCacheSave

```
ciCacheSave(
     u_cache
     )
     => t / nil
```

Description

Saves a constraint cellview.

Arguments

u_cache	Specifies the constraint cacheID of a schematic constraint cellview. See cache.

Value Returned

t	Constraint cellview successfully saved.
nil	Command failed.

Example

```
ciCacheSave(ciGetCellView())
```

Saves the constraint cache associated with the current window.

ciCacheTopCellName

```
ciCacheConstraintCellName(
     u_cache
     )
     => t_cellName
```

Description

Returns the top cell name of the specified constraint cache. The top library, cell, and view refers to the schematic or layout cellview. If the cache is associated with a configuration or physical configuration, it is the top cell specified by the configuration.

Arguments

u_cache	The constraint cache ID.

Value Returned

t_cellName	The top cell name of the associated schematic or layout.

Example

Return the top cell name of the current constraint cache:

```
ciCacheTopCellName(ciGetCellView())
=> "block1"
```

ciCacheTopLibName

```
ciCacheTopLibName(
     u_cache
     )
     => t_libName
```

Description

Returns the top library name of the specified constraint cache. The top library, cell, and view refers to the schematic or layout cellview. If the cache is associated with a configuration or physical configuration, it is the top cell specified by the configuration.

Arguments

u_cache	The constraint cache ID.

Value Returned

t_libName	The library name of the associated schematic or layout.

Example

Return the top library name of the current constraint cache:

```
ciCacheTopLibName(ciGetCellView())
=> "overview"
```

ciCacheTopViewName

```
ciCacheTopViewName(
     u_cache
     )
     => t_viewName
```

Description

Returns the top view name of the specified constraint cache. The top library, cell, and view refers to the schematic or layout cellview. If the cache is associated with a configuration or physical configuration, it is the top cell specified by the configuration.

Arguments

u_cache	The constraint cache ID.

Value Returned

t_viewName	The view name of the associated schematic or layout.

Example

Return the top view name of the current constraint cache:

```
ciCacheTopViewName(ciGetCellView())
=> "schematic"
```

or

```
ciCacheTopViewName(ciGetCellView())
=> "layout"
```

ciCacheTransfer

```
ciCacheTransfer(
     u_fromCache
     u_toCache
     ?conTypes s_conTypes
     )
     => t / nil
```

Description

Transfers constraints from the source cache to the specified target cache. The transfer can be restricted to constraints of a specified type. It cannot be used to transfer constraints between two schematics or two layouts.

Note: Before you run `ciCacheTransfer`, ensure that the schematic and layout windows are open in XL or a higher tier.

Arguments

u_fromCache	The source constraint cache ID from which the selected constraints are transferred.
u_toCache	The target constraint cache ID to which the constraints are transferred. The cache must be writeable.
`?conTypes` *s_conTypes*	The symbol for the type of constraint used to filter the specified list.

Note: The source and target constraint cache IDs must be bound together.

Value Returned

`t`	The selected constraints were transferred successfully.
`nil`	The transfer of the selected constraints failed.

Examples

Transfers all constraints from the schematic to the layout:

```
schCache = (ciGetCellView (window 2))
layCache = (ciGetCellView (window 3))
```

Transfers all constraints from the schematic cache to the layout cache:

```
(ciCacheTransfer schCache layCache)
```

Transfers all netClass constraints from the schematic cache to the layout cache:

```
(ciCacheTransfer schCache layCache ?conTypes 'netClass)
```

Transfer netClass and alignment constraints from the schematic cache to the layout cache:

```
(ciCacheTransfer schCache layCache ?conTypes '(netClass alignment))
```

ciCacheTransferSelection

```
ciCacheTransferSelection(
     u_fromCache
     u_toCache
     )
     => t / nil
```

Description

Transfers the selected constraints and templates from the source cache to the target cache in a session run in XL or above tier. If there are no selected constraints in the source cache, all the constraints and templates from the source cache are transferred to an editable target cache. When a constraint is transferred, the template within which it exists is also transferred along with all the other constraints inside the template.

Note: Before you run `ciCacheTransferSelection`, ensure that the schematic and layout windows are open in XL or above tier. The Constraint Manager assistant should also be open in the window from where the constraints need to be transferred.

Arguments

u_fromCache	The source constraint cache ID from where the selected constraints and templates need to be transferred. The specified cache ID should be valid. When an invalid cache ID is given, an error message is displayed.
u_toCache	The target constraint cache ID in which the selected constraints and templates need to be transferred.

Note: The source and target constraint cache IDs should be bound together. If this is not the case, the transfer fails with an error message. The `ciCacheTransferSelection` SKILL function cannot be used to transfer constraints between two schematics or two layouts.

Value Returned

`t`	Selected constraints and templates were transferred successfully.

`nil`	Transfer of the selected constraints and templates failed.

Examples

Consider that there are two caches bound together in an XL-tier session: `from` is the source cache and `to` is the target cache opened in edit mode.

The source cache is bound to a design for which *`libname`*, *`cellname`*, *`viewname`* are `'ether_adc45n'`, `'adc_cascode_opamp'`, and `'schematic'`.

The target cache is bound to the layout design associated to the source design.

```
;; For all constraints transfer
    ;;Deselect everything in all designs
    (let ((currentSelection (hsmGetSelectedSet)))
        (when currentSelection
        (hsmDeselect ?spec currentSelection)))
    ;;Transfer all constraints/templates from the source 'from' to the target 'to'
        (ciCacheTransferSelection from to)

;; For partial constraint transfer
    ;; Deselect everything in all design
    (let ((currentSelection (hsmGetSelectedSet)))
        (when currentSelection
        (hsmDeselect ?spec currentSelection)))
    ;;Select the first constraint found in the source cache
        (hsmSelect ?spec  `(("(ether_adc45n.adc_cascode_opamp:schematic)"
        (constraint ,(car from->constraints)))))
    ;;Transfer the selected constraint
        (ciCacheTransferSelection from to)
```

ciCacheViewName

```
ciCacheViewName(
     u_cache
     )
     => t_viewName
```

Description

Returns the view name of the specified constraint cache. The cache library, cell, and view refers to the schematic, configuration, physical configuration, or layout view that the constraint cache is associated with.

Arguments

u_cache	The constraint cache ID.

Value Returned

t_viewName	The view name of the associated schematic, configuration, physical configuration, or layout.

Example

Return the view name of the current constraint cache:

```
ciCacheViewName(ciGetCellView())
=> "physConfig"
```

ciCheckConstraints

```
ciCheckConstraints(
     d_cellViewID
     )
     => t / nil
```

Description

Runs the constraint checker on the passed schematic or layout cellview if the cellview contains constraints.

When the constraint checker is called for a schematic cellview, the constraint and template status checks start automatically based on the current settings in the *Constraints* tab of the Schematic Rules Check Setup form.

When the constraint checker is called for a layout cellview, the checks are performed through the PVS-CV licensed tool. Therefore, make sure that the `Phys_Ver_Sys_Const_Validator` (license number 96300) license has been checked out in addition to the Layout license. For more information, refer to Check Constraints (Layout) in *The Constraint Manager Assistant* chapter of the *Virtuoso Unified Custom Constraints User Guide.*

Arguments

d_cellViewID	The cellview identifier.

Value Returned

`t`	Constraint check successful.
`nil`	Command failed.

Example

```
ciCheckConstraints(geGetEditCellView())
```

ciCombineInstNetsPins

```
ciCombineInstNetsPins(
     l_instsNetsPins
     )
     => l_result
```

Description

This function is used within Circuit Prospector constraint generator expressions to turn a list of sub-lists into a single simple list. When the `instsNetsPins` information is represented as a list of sub-lists there is one sub-list each for instances, nets, pins, and instTerms. In addition, this function effectively flattens the sub-lists to leave a combined list of all the instances, nets, pins, and instTerms without sub-lists. The flattened list is in a form suitable to be used as the member list for ciConCreate.

To turn the result into a more manageable disembodied property list (DPL), see ciSeparateInstsNetsPins.

Arguments

l_instsNetsPins	Specifies the `instsNetsPins` as a list of sub-lists.

Value Returned

l_result	Returns a simple flattened list containing each instance, net, pin, and instTerm in the `instsNetsPins` sub-lists

Example

```
instsNetsPinsAsSubLists = '(("/MN5" "/MN2") ("/n5" "/gnd!") nil nil)

ciCombineInstNetsPins(instsNetsPinsAsSubLists)
     (("/MN5" 'inst)
     ("/MN2" 'inst)
     ("/n5" 'net)
     ("/gnd!" 'net))
```

ciConAppendOneMember

```
ciConAppendOneMember(
     u_constraint
     l_member
     )
     => t / nil
```

Description

Appends one constraint member to the given contstraint. The new member is added in the last position in the member list.

Arguments

u_constraint	Constraint user type ID.
l_member	Constraint member, represented as a list. The member is a pair (name type) or a triplet (name type parameter), listed in the same format as `ciConCreate`.

Value Returned

`t`	Member successful added.
`nil`	Member not added.

Example

```
ciConAppendOneMember(con '("MN12" inst))
```

Appends an extra instance, `MN12`, to a constraint.

ciConBaseName

```
ciConBaseName(
     t_constraintName
     )
     => baseName
```

Description

Extracts and returns the basename from the specified constraint name. A constraint name consists of the basename and the counter added as a suffix. For example, in the constraint name, `myConstraint_2`, the basename is `myConstraint` and the counter is `_2`.

Arguments

t_constraintName	The constraint name.

Value Returned

baseName	The constraint name without the counter suffix, that is, the constraint basename.

Example

```
cache = ciGetCellView()
conName = car(cache~>constraints~>name)
newUniqueName = ciNextConName(cache ciConBaseName(conName))
```

ciConCallbackIsRegistered

```
ciConCallbackIsRegistered(
     ?name t_name
     ?type t_type
     )
     => t / nil
```

Description

Checks whether a constraint callback has been registered.

See also ciConRegisterCallback, ciConUnregisterCallback, and ciConUpdateCallback.

Arguments

`?name` *t_name*	The name passed as a symbol of the callback.
`?type` *t_type*	The type passed as a symbol of the callback. Currently, `ciConTransferCallback` is the only supported type.

Value Returned

`t`	The specified callback is a registered callback.
`nil`	The specified callback is not a registered callback.

Examples

- If `myTransferCallback` is not registered as a callback, the function returns `nil`.

  ```
  ciConCallbackIsRegistered(?name 'myTransferCallback ?type
  'ciConTransferCallback)
  nil
  ```

- If `myTransferCallback` was registered as a callback of type `ciConTransferCallback` by using the `ciConRegisterCallback` function, the `ciConCallbackIsRegistered` function returns `t`. This means that the `myTransferCallback` callback will be called before and after the constraints are transferred.

  ```
  ;;Register the callback.
  ciConRegisterCallback(?name 'myTransferCallback ?type 'ciConTransferCallback)
  ```

```
t
;;Later, you can verify whether the callback was registered successfully.
ciConCallbackIsRegistered(?name 'myTransferCallback ?type
'ciConTransferCallback)
t
```

ciConCreate

```
ciConCreate(
     u_cache
     d_constraintType
     [ ?members l_memberList ]
     [ ?params l_parameterList ]
     [ ?name g_constraintName ]
     [ ?axis g_axisName ]
     [ ?note g_constraintNote ]
     [ ?verbose g_verbose ]
     )
     => u_constraint / nil
```

Description

Creates a new constraint.

Arguments

u_cache	See `cache`.
d_constraintType	A legal constraint type. See `constraint_type`.
?members *l_memberList*	An ordered constraint member list. See `member_list`.
?params *l_parameterList*	An optional list of constraint parameters. See `parameter_list`.
?name *g_constraintName*	An optional legal string specifying the constraint name. A constraint with that name must not already exist in the cache. If not specified, a name will be generated as `Constr_N`.
?axis *g_axisName*	An optional axis name to which the constraint belongs. When an axis with the given name does not exist it will be created.
?note *g_constraintNote*	An optional argument that enables you to specify notes to constraints to provide more details.
?verbose *g_verbose*	A Boolean argument that controls whether a message is displayed to inform of the successful creation of a constraint. Default: `t`

Value Returned

u_constraint	The created constraint. (see u_constraint).
nil	Constraint not created.

Examples

1. Creating a Symmetry constraint:

```
ciConCreate(
    cache 'symmetry
    ?members list(list("M1" 'inst) list("M2" 'inst))
    ?axis "vSym"
)
```

```
(If the specified axis does not exist then you might need to create it using
the ciAxisCreate() API).
```

2. Creating a Modgen constraint:

```
modgen=ciConCreate(
    cache 'modgen
    ?members list(list("M1" 'inst) list("M2" 'inst))
    ?params list(list("numRows" 1) list("numCols" 2)
    ))
```

In the example above, a Modgen constraint is generated with basic parameters. The parameters specify the Modgen member instances and the size of the array. Default values are used for parameters that are not specified, for example the symbol assignments.

Note: In Schematic, m-factored instances are not expanded. Therefore, the parameters specified pertain to the unexpanded values, such as rows and columns.

You can generate a Modgen constraint with member parameters (in Layout):

```
?members
list(
list("M1.1" 'inst list(list("row" 0) list("col" 0) list("abutment" 1)))
list("M1.2" 'inst list(list("row" 0) list("col" 1) list("abutment" 1)))
list("M2.1" 'inst list(list("row" 0) list("col" 2) list("abutment" 1)))
list("M2.2" 'inst list(list("row" 0) list("col" 3) list("abutment" 1)))
)
```

As specified above, m-factored instances are not expanded in the schematic, which makes it difficult to specify member parameters for the expanded instances. However, you may specify multiple instances in such cases as follows:

```
list(
list("M1" `inst list(list("row" 0) list("col" 0) list("abutment" 1)))
list("M1" `inst list(list("row" 0) list("col" 1) list("abutment" 1)))
list("M2" `inst list(list("row" 0) list("col" 2) list("abutment" 1)))
list("M2" `inst list(list("row" 0) list("col" 3) list("abutment" 1)))
)
```

Here, both `M1` and `M2` have an m-factor of `2`. Also, the row and column assignment for each instance is given.

You can also specify Modgens with dummy devices and associated parameters (in schematic):

```
memberList = list(
list("gpdk045/nmos2v/layout" master (("row" 0) ("col" 0) ("abutment" 1)
list("dummyNetToUse" 1) ("dummyParams" "((fingers 1) (fw 4u) (l 400n))")))
list("M3" inst (("row" 0) ("col" 1) ("abutment" 1)))
list("M2" inst (("row" 0) ("col" 2) ("abutment" 1)))
list("M2" inst (("row" 0) ("col" 3) ("abutment" 1)))
list("M3" inst (("row" 0) ("col" 4) ("abutment" 1)))
list("gpdk045/nmos2v/layout" master (("row" 0) ("col" 5) ("abutment" 1)
list("dummyNetToUse" 1) ("dummyParams" "((fingers 1) (fw 4u) (l 400n))")))
list("gpdk045/nmos2v/layout" master (("row" 1) ("col" 0) ("abutment" 1)
list("dummyNetToUse" 1) ("dummyParams" "((fingers 1) (fw 4u) (l 400n))")))
list("M2" inst (("row" 1) ("col" 1) ("abutment" 1)))
list("M3" inst (("row" 1) ("col" 2) ("abutment" 1)))
list("M3" inst (("row" 1) ("col" 3) ("abutment" 1)))
list("M2" inst (("row" 1) ("col" 4) ("abutment" 1)))
list("gpdk045/nmos2v/layout" master (("row" 1) ("col" 5) ("abutment" 1)
list("dummyNetToUse" 1) ("dummyParams" "((fingers 1) (fw 4u) (l 400n))")))
list("vdd!" net) ;;; dummy net
)


modgen2 = ciConCreate(
cache 'modgen
?members memberList
?params list(
list("numRows" 2)
list("numCols" 6)

list("pattern" "mapping (( M2 X)  (M3 Y))")
    ); list)
)
```

In the above example, the pattern parameter is used to map symbols to alphabets.

```
ciConCreate( d_cache  'modgen
?members list(
list("R0" 'inst list( list("row" 5 ) list("col" 0 ) list("orient" "R0" )))
list("R0" 'inst list( list("row" 4 ) list("col" 0 ) list("orient" "MY" )))
list("R0" 'inst list( list("row" 1 ) list("col" 0 ) list("orient" "R0" )))
list("R1" 'inst list( list("row" 0 ) list("col" 0 ) list("orient" "MY" )))
list("R1" 'inst list( list("row" 2 ) list("col" 0 ) list("orient" "R0" )))
list("R1" 'inst list( list("row" 3 ) list("col" 0 ) list("orient" "MY" )))
);list
?params list(
list( "numRows" 6 )
list( "numCols" 1 )

);ciConCreate
```

As shown in the example above, you can specify members and parameters for deciding the placement order of the Modgen devices that are generated from schematic and where both `R0` and `R1` have an s-factor of `3`.

As shown in the example below, to add a dummy that has a master different from its neighbor, specify the additional `dummyParamSource` parameter.

```
("gpdk045/nmos2v_3/layout" master (("row" 0) ("col" 11) ("abutment" 1)
("dummyParamSource" "default")
```

To specify dummy parameters, use the additional `dummyParams` parameter, as shown below:

```
("gpdk045/nmos2v_3/layout" master (("row" 0) ("col" 11) ("abutment" 1)
("dummyParamSource" "default")  ("dummyParams" "((fingers 1) (fw 4u) (l
400n))")))
```

If specifying a dummy different from the neighbor, the `dummyParams` should be specified. Otherwise, the values of the dummy parameters (length, width, fingers etc.) will be set to minimum.

3. Creating a constraint using another existing constraint as its member:

Suppose there are three instances outside of modgens, and three modgens (each having some number of instances). If you want to create an orientation constraint of only six objects, which includes these three instances and three modgens, you can use a syntax like the one shown below:

```
ciConCreate(
    cache ?type orientation
    ?members list( list("M1" 'inst) list("Modgen1" 'modgen))
    ?params list(list("restrictTo" list("R0" "MY")))
)
```

Note: This type of constraint creation is supported for the objects listed in the design_object_type section.

ciConCreateExpanded

```
ciConCreateExpanded(
     u_cache
     d_constraintType
     [ ?members l_memberList]
     [ ?params l_parameterList ]
     [ ?name t_constraintName ]
     [ ?axis t_axisName ]
     [ ?note g_constraintNote ]
     [ ?verbose g_verbose ]
     )
     => l_constraints / nil
```

Description

Expands vector names into groups of vector bit names in a constraint specific manner (when a constraint's member list contains vector names). Constraints are then created for each vector bit name grouping.

Note: When the member list does not contain vector names this function will behave the same as ciConCreate, except that it will return a list with a single constraint rather than the constraint itself.

Arguments

u_cache	See `cache`.
d_constraintType	A legal constraint type. See `constraint_type`.
?members *l_memberList*	An ordered constraint member list. See `member_list`.
?params *l_parameterList*	An optional list of constraint parameters. See `parameter_list`.
?name *t_constraintName*	An optional legal string that is unique in the constraint naming cache. If not specified, a name will be generated automatically.
?axis *t_axisName*	An optional axis name to which the constraint belongs. When an axis with the given name does not exist it will be created.
?note *g_constraintNote*	An optional argument that enables you to specify notes to constraints to provide more details.

`?verbose` *`g_verbose`*	A Boolean argument that controls whether a message is displayed to inform of the successful creation of a constraint. Default: `t`

Value Returned

`l_constraints`	A list of the newly-created constraints.
`nil`	Constraint not created.

Example

```
ciConCreateExpanded(cache 'symmetry ?members list(list("MN1<0:4>" 'inst)))
=> list( symmetry("MN1<0>", "MN1<4>"), symmetry("MN1<1>", "MN1<3>"),
symmetry("MN1<2>")

ciConCreateExpanded(cache 'cluster ?members list(list("MN1<0:4>" 'inst)))
=> list( cluster("MN1<0>", "MN1<1>", "MN1<2>", "MN1<3>", "MN1<4>")
```

ciConDelete

```
ciConDelete(
     u_constraint
     )
     => t / nil
```

Description

Deletes a constraint. After deleting the constraint, the `u_constraint` will be invalid.

Note: Using the `constraint_id` after the original constraint has been deleted can cause fatal errors.

Arguments

u_constraint	The constraint to be deleted.

Value Returned

`t`	Constraint successfully deleted.
`nil`	Constraint not deleted.

ciConFind

```
ciConFind(
     u_cache
     t_constraintName
     )
     => u_constraint / nil
```

Description

Finds a constraint in a given `cache`.

Arguments

u_cache	The constraint cache in which the constraint belongs. See cache.
t_constraintName	The name of the constraint to be found.

Value Returned

u_constraint	The `u_constraint` of the found constraint.
`nil`	No constraint found.

ciConGetAxisName

```
ciConGetAxisName(
     u_constraint
     )
     => t_axisName / nil
```

Description

Returns, for a given constraint, the axis name if one is associated with the given constraint.

Arguments

u_constraint	The id of the constraint to obtain an associated axis name from (see `u_constraint`).

Value Returned

t_axisName	The name of the axis found.
`nil`	No axis found.

ciConGetCache

```
ciConGetCache(
     u_constraint
     )
     => cache_ID
```

Description

Returns the constraint cache that contains the given constraint.

Arguments

u_constraint	The ID of the constraint (see u_constraint).

Value Returned

cache_ID	The ID of the constraint cache.

Example

```
ciConGetCache(conId) => cacheId
```

ciConGetComment

```
ciConGetComment(
     u_constraint
     )
     => t_comment / nil
```

Description

Returns the comment parameter of a constraint.

Note: A comment is associated with a constraint's status.

Arguments

u_constraint	The id of the constraint to obtain details of any comment parameters from (see u_constraint).

Value Returned

t_comment	Details of any comment found.
nil	No comment found.

ciConGetCreatedTime

```
ciConGetCreatedTime(
     u_constraint
     )
     => constraint_created_time / nil
```

Description

Returns the created time of a constraint.

Note: The constraint ID must be a legal reference to a constraint. Using a constraint ID after a constraint has been deleted can result in a fatal error.

Arguments

u_constraint	The id of the constraint whose creation time you want to return (see u_constraint).

Value Returned

constraint_created_time	The created time of the given constraint.
nil	No constraint of given name found.

ciConGetMembersOfType

```
ciConGetMembersOfType(
     g_con
     s_type
     )
     => l_filteredList
```

Description

Returns the constraint member list filtered by the passed member type.

Arguments

g_con	The constraint for which the filtered member list should be returned.
s_type	The constraint member type to filter the member list by.

Value Returned

l_filteredList	A filtered version of the constraint members only containing members of the specified type.

Example

```
ciConGetMembersOfType(modgen 'inst)
```

Returns the instance members of the modgen, but will not return the net members (if the modgen has members of type net).

ciConGetName

```
ciConGetName(
     u_constraint
     )
     => t_constraintName
```

Description

Returns the name of a constraint.

Note: The `u_constraint` must be a legal reference to a constraint. Using a `constraint_id` after a constraint has been deleted can result in a fatal error.

Arguments

u_constraint	The ID of the constraint whose name you want to return.

Value Returned

t_constraintName	The name of the constraint.

ciConGetNote

```
ciConGetNote(
     u_constraint
     )
     => t_note / nil
```

Description

Returns the note parameter of a constraint.

Arguments

`u_constraint`	The ID of the constraint whose note you want to return (see `u_constraint`).

Value Returned

`t_note`	Details of the note if it exists for the given constraint.
`nil`	No constraint note found.

ciConGetOwner

```
ciConGetOwner(
     u_constraint
     )
     => l_lcv / nil
```

Description

Returns the library name, cell name, and view name of the cell where the passed constraint was created.

Arguments

u_constraint	The ID of the constraint whose library, cell, and view origin information you want to return (see u_constraint).

Value Returned

l_lcv	Lists library, cell, and view name of the passed constraint.
nil	Command failed.

Example

```
ciConGetOwner(u_constraint)
=> ("amsPLL" "vco" "schematic")
```

ciConGetPriority

```
ciConGetPriority(
     u_constraint
     )
     => x_priority
```

Description

Returns the priority value of a constraint.

Arguments

`u_constraint`	The ID of the constraint whose priority value you want to return (see `u_constraint`).

Value Returned

`x_priority`	The priority value of the given constraint. It is an integer between `[0...255]`.

ciConGetStatus

```
ciConGetStatus(
     u_constraint
     )
     => symbol / nil
```

Description

Returns the status of a constraint.

See also ciConSetStatus.

Arguments

u_constraint	The ID of the constraint whose status you want to return (see `u_constraint`).

Value Returned

`symbol: violated`	Constraint status is current set to `violated`
`symbol: enforced`	Constraint status is currently set to `enforced`.
`symbol: impossible`	Constraint status is currently set to `disabled`.
`nil`	No constraint status found.
	A `nil` status is referrred to as “not checked”.

Example

```
ciConSetStatus(cc 'disabled t)
ciConGetStatus(cc)
disabled
```

ciConGetType

```
ciConGetType(
     u_constraint
     )
     => s_constraintTypeName
```

Description

Returns the constraint type symbol.

Arguments

u_constraint	The ID of the constraint whose type symbol you want to return.

Value Returned

s_constraintTypeName	Details of the constraint_type symbol.

Example

```
ciConSetStatus(cc 'disabled t)
```

ciConIsInContext

```
ciConIsInContext(
     u_constraint
     )
     => t / nil
```

Description

Returns the current context status of a constraint.

Note: Being “in context” refers to design information, such as a constraint, being displayed in the current window/view.

Arguments

u_constraint	The ID of the constraint whose context status you want to return (see u_constraint).

Value Returned

t	All constraint members are in context.
nil	One or more members current constraint status is not in context.

ciConIsOutOfContext

```
ciConIsOutOfContext(
     u_constraint
     )
     => t / nil
```

Description

Returns the current of context status of a constraint.

Note: Being "out of context" refers to design information, such as a constraint, not being displayed in the current window/view.

Arguments

u_constraint	The ID of the constraint whose context status you want to return (see u_constraint).

Value Returned

t	All constraint members are out of context.
nil	One or more members current constraint status is not out of context.

ciConIsOverridden

```
ciConIsOverridden(
     u_constraint
     )
     => t / nil
```

Description

Returns whether a constraint has been overridden.

Note: Constraint overrides provide the option of having a constraint set as overridden rather than deleted or modified. Setting a constraint as overridden can allow, for example, layout constraints to differ from schematic constraints and also for variants of constraints to exist.

Arguments

u_constraint	The ID of the constraint that you want to establish has been overridden or not (see `u_constraint`).

Value Returned

`t`	Constraint has been overridden.
`nil`	Constraint has not been overridden.

ciConIsWritable

```
ciConIsWritable(
     u_constraint
     )
     => t / nil
```

Description

Returns whether the constraint is writable.

Note: To ensure that a constraint is writable, the constraint view must be editable.

Arguments

u_constraint	The ID of the constraint that you want to establish writable status (see `u_constraint`).

Value Returned

`t`	Constraint is writable.
`nil`	Constraint is read-only.

ciConListMembers

```
ciConListMembers(
     u_constraint
     [ g_includeParameters ]
     [ g_includeDefaults ]
     )
     => l_memberList / nil
```

Description

Returns a list of constraint members and their member parameters, if any.

Arguments

u_constraint	The constraint ID (see `u_constraint`).
g_includeParmeters	Optional argument. A Boolean value (`t` if unspecified) that includes or excludes parameters. If argument is set as `nil`, the parameter_list component of the constraint_member will not be returned.
g_includeDefaults	Optional argument. A Boolean value (`t` if unspecified) that includes or excludes parameters with default values when `t` or `nil` respectively.

Value Returned

l_memberList	List of constraint members name and types (see `member_list`).
`nil`	No constraint members found for constraint.

Example

```
(ciConListMembers con) ;;
```

Includes all parameters, including defaults.

```
(ciConListMembers con nil) ;;
```

Do not include parameters.

ciConListMemberNames

```
ciConListMemberNames(
     u_constraint
     )
     => l_memberList / nil
```

Description

Returns a list of strings where each string is a constraint member name. This function will only return member names as opposed to ciConListMembers which returns tuples containing member names and types.

Arguments

u_constraint	The `u_constraint`.

Value Returned

l_memberList	List of constraint members names (see `member_list`).
`nil`	No constraint ID found for constraint.

ciConListParams

```
ciConListParams(
     u_constraint
     [ g_includeDefaults ]
     )
     => l_parameterList / nil
```

Description

Returns all parameters for a constraint.

Arguments

u_constraint	The `u_constraint` whose parameters you want to list.
g_includeDefaults	Optional argument. A Boolean value (`t` if unspecified) that includes or excludes parameters with default values when `t` or `nil` respectively.

Value Returned

l_parameterList	List of parameters found for given constraint (see `parameter_list`).
`nil`	No parameters found for constraint.

ciConListParamNames

```
ciConListParamNames(
     u_constraint
     )
     => l_parameterNames / nil
```

Description

Returns the legal parameter names for a given constraint.

Arguments

u_constraint	The constraint ID whose parameters you want to list (see u_constraint).

Value Returned

l_parameterNames	List of parameter names returned for the given constraint.
`nil`	No parameters found for the given constraint.

ciConListTemplates

```
ciConListTemplates(
     u_constraint
     )
     => l_templateList / nil
```

Description

Returns the list of templates of which the given constraint is a member.

Arguments

u_constraint	The constraint ID whose templates you want to list (see u_constraint).

Value Returned

l_templateList	List of templates returned for the given constraint.
nil	No templates found for the given constraint.

Example

```
con = ciConFind(cache, "Const_1")
tempList = ciConListTemplates(con)
```

ciConp

```
ciConp(
     g_value
     )
     => t / nil
```

Description

Checks if an object is a valid constraint ID.

Arguments

g_value	Specifies the object to be checked.

Value Returned

`t`	If *g_value* is a valid constraint ID.
`nil`	If *g_value* is not a valid constraint ID.

Example

```
ciConp(
     obj
     )
=>t
```

As shown in the above example, the object, `obj` is a valid constraint.

ciConRegisterCallback

```
ciConRegisterCallback(
     ?name t_name
     ?type t_type
     )
     => t / nil
```

Description

Registers a constraint callback.

See also ciConCallbackIsRegistered, ciConUnregisterCallback, and ciConUpdateCallback.

Arguments

`?name` *`t_name`*

The name passed as a symbol of the callback procedure to be registered.

`?type` *`t_type`*

The type passed as a symbol of the callback. Currently, `ciConTransferCallback` is the only supported type.

A callback of the type `ciConTransferCallback` is called before and after the transfer of constraints. Such a callback accepts the following arguments:

- `scx`: The schematic cache.
- `lcx`: The layout cache.
- `directionL2P`: Gives the direction of the transfer. If the transfer occurs from the logical cellview (that is, schematic) to the physical cellview (that is, layout), `directionL2P` is set to `t`; otherwise, it is set to `nil`.
- `status`: Can be one of the following symbols: `'preTransfer` or `'postTransfer`.

Value Returned

`t`	The callback was registered successfully.
`nil`	The callback could not be registered.

Example

```
procedure(myTransferCallback(scx lcx directionL2P status)
;;print a message each time the callback is called
  printf("myTransferCallback: scx %L - lcx %L - directionL2P %L - status %L\n" scx
lcx directionL2P status)
)

;;Register the "myTransferCallback" as callback of type 'ciConTransferCallback.
;;"myTransferCallback" will be called before and after constraints are transferred.
ciConRegisterCallback(?name 'myTransferCallback ?type 'ciConTransferCallback)
t
```

ciConRemoveMembers

```
ciConRemoveMembers(
     u_constraint
     l_memberList
     )
     => l_memberList / nil
```

Description

Removes a list of members from a constraint.

Arguments

u_constraint	The u_constraint whose members you want to remove.
l_memberList	The list of members to be removed. See member_list.

Value Returned

l_memberList	List of members removed for given constraint.
nil	No members removed from constraint.

Example

```
ciConListMembers(a)
(("PM0" inst nil)
    ("PM6" inst nil)
    ("PM4" inst nil)
    ("PM1" inst nil)
    ("PM2" inst nil)
)
>
ciConRemoveMembers(a list(list("C3" 'inst) list("C4" 'inst)))
t
>
ciConListMembers(a)
(("PM0" inst nil)
    ("PM6" inst nil)
```

```
    ("PM4" inst nil)
    ("PM1" inst nil)
    ("PM2" inst nil)
)
```

ciConResetAllParams

```
ciConResetAllParams(
     u_constraint
     )
     => t / nil
```

Description

Resets all parameters on a constraint to their default values. Parameters can only be reset if the constraint view is currently writable.

Arguments

u_constraint	The `u_constraint` whose parameter values you want to reset.

Value Returned

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

ciConResetParams

```
ciConResetParams(
     u_constraint
     [ l_parameterNameList ]
     )
     => t / nil
```

Description

Resets a list of `parameter`s to their default values.

Arguments

u_constraint	The `u_constraint` whose parameter values you want to reset.
l_parameterNameList	List of parameter names to be reset to their default values. See parameter_list.

Value Returned

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

ciConSetAxis

```
ciConSetAxis(
     u_constraint
     t_axisName
     )
     => t / nil
```

Description

Associates a given constraint with the named axis. If the constraint does not accept an axis, `nil` is returned, and if the axis does not exist it will be created.

Arguments

u_constraint	The identifier for the constraint that you want to associate with a given axis (see also `u_constraint`).
t_axisName	The axis name. If the axis does not exist one will be created with the default `parameter`s. Setting to `nil` will remove the axis from the constraint.

Value Returned

`t`	Successful association of constraint with given axis.
`nil`	Constraint does not allow an axis or another failure has occurred.

ciConSetNote

```
ciConSetNote(
     u_constraint
     t_note
     )
     => t / nil
```

Description

Replaces the note parameter of a constraint.

Arguments

u_constraint	The u_constraint whose note you want to replace.
t_note	The new note string.

Value Returned

t	Note successfully replaced.
nil	Note not replaced.

ciConSetPriority

```
ciConSetPriority(
     u_constraint
     x_priorityValue
     )
     => t / nil
```

Description

Sets the priority for a constraint.

Arguments

u_constraint	The `u_constraint` whose priority you want to set.
x_priorityValue	Sets the priority value for the constraint. It is an integer value between `[0...255]`.

Value Returned

`t`	Priority successfully set.
`nil`	Priority not set.

ciConSetStatus

```
ciConSetStatus(
     u_constraint
     t_statusSymbol
     g_statusValue
     )
     => t / nil
```

Description

Sets a given status flag for a constraint.

See also ciConGetStatus.

Arguments

u_constraint	The `u_constraint`.
t_statusSymbol	The status symbol can be either: ‘`violated`, ‘`enforced`, or ‘`disabled`. **Note:** `'impossible` is also supported for compatibility purposes and is treated the same as `'violated`.
g_statusValue	The status value: `t` or `nil`. Specifying `nil`, with any of the status symbols, will clear the constraint status to `nil`. A `nil` status is referred to as “not checked”.

Value Returned

`t`	Status successfully set for constraint.
`nil`	Status not set.

Example

```
ciConSetStatus(cc 'disabled t)
ciConGetStatus(cc)
disabled
```

ciConstraintsForType

```
ciConstraintsForType(
     l_constraints
     s_conType
     )
     => l_filteredConstraints / nil
```

Description

Filters the list of constraints and returns the constraints of the specified type.

Arguments

l_constraints	The list of constraints to be filtered.
s_conType	The symbol for the type of constraint to filter the specified list with.

Value Returned

l_filteredConstraints	The filtered list of constraints.
nil	Failed to filter the specified type of constraints.

Example

```
ciConstraintsForType(template->constraints 'modgen)~>name
=> '("modgen1")

ciConstraintsForType(ciGetCellView()->constraints 'modgen)~>name
=> '("modgen1" "modgen2" "modgen3")
```

ciConstraintLCV

```
ciConstraintLCV(
     u_cache
     )
     => l_libCellViewNames
```

Description

Returns a list containing the library, cell, and view names of the specified constraint cache. The constraint library, cell, and view refers to the storage location of the constraints, which may be a constraint view or layout.

Arguments

u_cache	The constraint cache ID.

Value Returned

l_libCellViewNames	List comprising the storage library, cell and view names of the constraint view or layout returned in the following format:
	`( t_libName t_cellName t_viewName )`
	For example:
	`( "overview" "block1" "constrBlock" )`

Example

Get the constraint storage library, cell, and view names of the current constraint cache:

```
ciConstraintLCV(ciGetCellView())
=> ("overview" "block1" "constrBlock")
```

or

```
ciConstraintLCV(ciGetCellView())
=> ("overview" "block1" "layout")
```

ciConstraintViewLessp

```
ciConstraintViewLessp(
     u_cache1 u_cache2 ...
     )
     => t / nil
```

Description

Specifies a predicate function for sorting the constraint caches based on their library, cell, and view names.

Arguments

u_cache1 u_cache2 ...	A list of constraint caches.

Value Returned

`t`	Constraint caches sorted successfully.
`nil`	Constraint caches could not be sorted.

Example

The following sorts a list of read-only constraint caches:

```
readonly = setof(c ciGetOpenCellViews() null(ciCacheIsWritable(c)))
mapcar(lambda( (c) info("Cache %L constraint view %L\n" ciCacheLCV(c)
ciConstraintLCV(c))) sort(readonly 'ciConstraintViewLessp))
Cache ("overview" "block1" "physConfig") constraint view ("overview" "block1"
"constrBlock")
Cache ("overview" "block2" "schematic") constraint view ("overview" "block2"
"constraint")
Cache ("overview" "upper" "schematic") constraint view ("overview" "upper"
"constraint")
```

ciConTypeHasNamedParameter

```
ciConTypeHasNamedParameter(
     t_conType
     t_paramName
     )
     => t / nil
```

Description

Returns `t` if the passed constraint type has a parameter with a name matching the passed parameter name.

Arguments

t_conType	The constraint type name
t_paramName	The constraint parameter name.

Value Returned

`t`	The parameter name of the passed constraint type matches the passed parameter name.
`nil`	Command failed.

Examples

```
ciConTypeHasNamedParameter( "modgen" "numRows")
=> t
```

```
ciConTypeHasNamedParameter( "relativeOrientation" "axis")
=> nil
```

ciConUnregisterCallback

```
ciConUnregisterCallback(
     ?name t_name
     ?type t_type
     )
     => t / nil
```

Description

Unregisters a constraint callback.

See also ciConCallbackIsRegistered, ciConRegisterCallback, and ciConUpdateCallback.

Arguments

`?name` *`t_name`*	The name passed as a symbol of the callback procedure to be unregistered.
`?type` *`t_type`*	The type passed as a symbol of the callback. Currently, `ciConTransferCallback` is the only supported type.

Value Returned

`t`	The callback was unregistered successfully.
`nil`	The callback could not be unregistered.

Example

```
procedure(myTransferCallback(scx lcx directionL2P status)
;;print a message each time the callback is called
  printf("myTransferCallback: scx %L - lcx %L - directionL2P %L - status %L\n" scx
lcx directionL2P status)
)

;;function is not yet registered.So, the function won't be unregistered.
ciConRegisterCallback(?name 'myTransferCallback ?type 'ciConTransferCallback)
nil

;;Register the "myTransferCallback" as callback of type 'ciConTransferCallback.
```

```
;;"myTransferCallback" will be called before and after constraints are transferred.
ciConRegisterCallback(?name 'myTransferCallback ?type 'ciConTransferCallback)
t

;;Now the function can be unregistered
ciConRegisterCallback(?name 'myTransferCallback ?type 'ciConTransferCallback)
t
```

ciConUpdateCallback

```
ciConUpdateCallback(
     ?name t_name
     ?type t_type
     [ ?enabled g_enabled ]
     )
     => t / nil
```

Description

Enables or disables a constraint callback.

See also ciConCallbackIsRegistered, ciConRegisterCallback, and ciConUnregisterCallback.

Arguments

`?name` *t_name*	The name passed as a symbol of the callback procedure to be unregistered.
`?type` *t_type*	The type passed as a symbol of the callback. Currently, `ciConTransferCallback` is the only supported type.
`?enabled` *g_enabled*	A Boolean value to specify whether the callback should be enabled. If set to `t`, the callback is enabled; otherwise, the callback is disabled and will not be called. By default, this argument is set to `t`. **Note:** This `@key` argument is optional.

Value Returned

`t`	The callback was updated successfully.
`nil`	The callback could not be updated.

Example

```
procedure(myTransferCallback(scx lcx directionL2P status)
;;print a message each time the callback is called
  printf("myTransferCallback: scx %L - lcx %L - directionL2P %L - status %L\n" scx
lcx directionL2P status)
```

```
)

;;function is not yet registered.So, the function will be registered.
ciConUpdateCallback(?name 'myTransferCallback ?type 'ciConTransferCallback
?enabled t)
t

ciConCallbackIsRegistered(?name 'myTransferCallback ?type 'ciConTransferCallback)
t

;;function is registered and enabled. So nothing is done.
ciConUpdateCallback(?name 'myTransferCallback ?type 'ciConTransferCallback
?enabled t)
nil

;;function is registered and is of type 'ciConTransferCallback. So, the function
will be unregistered.
ciConUpdateCallback(?name 'myTransferCallback ?type 'ciConTransferCallback
?enabled nil)
t

ciConCallbackIsRegistered(?name 'myTransferCallback ?type 'ciConTransferCallback)
nil
```

ciConUpdateMemberParams

```
ciConUpdateMemberParams(
     g_conId
     l_memberParams
     )
     => t / nil
```

Description

Updates the specified constraint member parameters with the given values without impacting other member parameters.

Arguments

g_conId	The ID of the constraint whose member parameters are to be updated.
l_memberParams	The list of constraint members and parameters to be updated.

Value Returned

`t`	The specified member parameters were successfully updated.
`nil`	No member parameters were updated. For example, if all the member parameter values are already set to the passed values, the constraint is not updated.

Example

Updates the member parameters associated with the high current design intent `HighCurrent_VDD`. The Current parameter is changed from `-5.0` to `-11.0` for VDD:2 and from `0.0` to `11.0` for M12:S. The parameters of the other members associated with the design intent are unchanged.

```
cache = ciGetCellView()
conId = car(ciTemplateFind(cache "HighCurrent_VDD")->constraints)
foreach(mapcar mbr conId->members list(car(mbr) assoc("Current" caddr(mbr))))
(("VDD:2"
    ("Current" float -5.0)
    )
```

```
    ("PM0:S"
    ("Current" float 1.0)
    )
    ("PM7:S"
    ("Current" float 2.0)
    )
    ("M5:S"
    ("Current" float 2.0)
    )
    ("PM2:S"
    ("Current" float 0.0)
    )
    ("M4:S"
    ("Current" float 0.0)
    )
    ("M12:S"
    ("Current" float 0.0)
    )
    ("PM1<0>:S"
    ("Current" float 0.0)
     )
     ("PM1<1>:S"
    ("Current" float 0.0)
    )
)


ciConUpdateMemberParams(conId list( list("VDD:2" list(list("Current" -11.0)))
list("M12:S" list(list("Current" 11.0)))))

foreach(mapcar mbr conId->members list(car(mbr) assoc("Current" caddr(mbr))))
(("VDD:2"
        ("Current" float -11.0)
    )
    ("PM0:S"
        ("Current" float 1.0)
    )
    ("PM7:S"
        ("Current" float 2.0)
    )
    ("M5:S"
        ("Current" float 2.0)
    )
    ("PM2:S"
        ("Current" float 0.0)
    )
    ("M4:S"
        ("Current" float 0.0)
    )
    ("M12:S"
        ("Current" float 11.0)
    )
    ("PM1<0>:S"
        ("Current" float 0.0)
    )
    ("PM1<1>:S"
        ("Current" float 0.0)
    )
)
```

ciConUpdateMembers

```
ciConUpdateMembers(
     u_constraint
     l_memberList
     )
     => t / nil
```

Description

Used to update members and their parameters by replacing the members with the list contained in `l_memberList`.

The list can be of the form `(("PM1" inst) ("PM2" inst))`, but not `(("PM1" inst nil) ("PM2" inst nil))`. That is, if there are no parameters, the parameters element in the tuples should be omitted.

Note: Reordering of members is not possible and should be done in conjunction with `ciConRemoveMembers`.

Arguments

u_constraint	The `u_constraint` whose members and parameters you want to update.
l_memberList	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

The following example details how to include additional members on a cluster constraint using the `ciConUpdateMembers` command:

```
procedure(doClusterBoundaries(cv cache name align)
  let((conname1 conname2)
    printf("name in doClusterBoundariess is %L\n" name)
    if(align then
      if((conname1 = ciConFind(cache name)) then
    printf("adding in doClusterBoundariess is %L\n" align)

        ciConUpdateMembers(conname1 append(align foreach(mapcar x
ciConListMembers(conname1) list(car(x) cadr(x)))))

      else
    printf("creating in doClusterBoundariess is %L\n" align)
        conname1 = ciConCreate(cache 'cluster
                    ?members align
                    ?name name
        )
        ciConCreate(cache 'clusterBoundaryDef
                  ?members list(list(name 'cluster))
                  ?params  list(list("boundary"  x=getBoundary(cv "upper"))
list("flexibleFlag" 1))
        )
      )
    )
    printf("names = %L\n" ciConListMembers(conname1))
  )
)
```

ciConUpdateParams

```
ciConUpdateParams(
     u_constraint
     [ l_parameterList ]
     )
     => t / nil
```

Description

Updates parameter values with those values listed in `l_parameterList`.

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 appended.

Arguments

u_constraint	The `u_constraint` whose parameter values you want to update.
l_parameterList	The list of parameters to be used for the parameter update. See `parameter_list`.

Value Returned

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

ciConUprevCellBoundary

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

Description

Converts deprecated `boundaryArea` and `areaUtilization` constraints to `cellBoundary` constraints.

Arguments

d_cellView	The name of the cellview.

Value Returned

`t`	Successfully converted old constraints.
`nil`	No `boundaryArea` or `areaUtilization` constraints were found.

ciConVerify

```
ciConVerify(
     { u_cache [ d_constraintType ] } | { u_constraint }
     )
     => t / nil
```

Description

Runs the consistency checker on demand. The behavior of the consistency checker is based on the values provided for each argument.

Arguments

u_cache	See `cache` for the syntax of this argument. If this argument is specified without any other argument, the consistency checker runs on all constraints in the cache.
d_constraintType	A valid constraint type. See `constraint_type`. If this argument is specified with *u_cache*, the consistency checker runs on all constraints of the given type in the specified cache.
u_constraint	Specify the SKILL user-type object that references a constraint. The consistency checker is run on this constraint. To know the value of `u_constraint`, you can use ciConFind as illustrated below in the Examples section.

Value Returned

`t`	All consistency checks passed on all constraints.
`nil`	Consistency checks failed on at least one constraint.

Examples

Based on the above explanation, there are the following three possibilities that have been explained using examples:

- You specified the `ciCache` pointer or `(list "lib" "cell" "view")`, as illustrated below.

  ```
  ciConVerify(ciGetCellView())
  ```

  Here, `ciGetCellView()` returns the cache.

  OR

  ```
  ciConVerify('("lib" "cell" "view"))
  ```

  In this case, the consistency checker runs on all constraints in the cache.

- You specified the `ciCache` pointer or `(list "lib" "cell" "view")`, and the type name, as illustrated below.

  ```
  ciConVerify(ciGetCellView(), 'alignment)
  ```

  In this case, the consistency checker runs on all constraints of the specified type in the specified cache.

- You specified only the constraint pointer, as illustrated below.

  ```
  con = ciConFind(ciGetCellView(), "Constr_0")
  ciConVerify(con)
  ```

  In this case, the consistency checker runs on the specified constraint.

ciConvertNestedNetClassToNetClassHierGroup

```
ciConvertNestedNetClassToNetClassHierGroup(
     g_cacheId
     [ S_constraintName ]
     )
     => t / nil
```

Description

Converts one or all nested Net Class constraints in a cache to corresponding Net Class Hier Group constraints. The name, members, and parameters of the original nested Net Class constraint remain the same after the conversion.

Arguments

g_cacheId	Cache ID in which the conversion is required.
S_constraintName	Constraint name if only one conversion is required; otherwise, all the nested Net Class constraints in the cache are converted.

Value Returned

`t`	All nested Net Class constraints in the specified cache were converted to corresponding Net Class Hier Group constraints.
`nil`	Conversion of at least one nested Net Class constraint in the specified cache failed.

Examples

- Converts all nested Net Class constraints to Net Class Hier Group constraints:

  ```
  ciConvertNestedNetClassToNetClassHierGroup(ciGetCellView())
  ```

- Converts only one nested Net Class constraint named `Constr_1` to Net Class Hier Group constraint:

  ```
  ciConvertNestedNetClassToNetClassHierGroup(ciGetCellView() "Constr_1")
  ```

ciCreateFilter

```
ciCreateFilter(
     t_filterName
     l_data
     )
     => t / nil
```

Description

Defines a filter to show/hide constraints in the *Constraint Manager*.

For more information see Constraint Filters.

Arguments

t_filterName	The name of the constraint filter you want to create.
l_data	The constraint filter criteria that you want to apply.

Value Returned

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

Example

```
ciCreateFilter( "MyAlign"
    list(
        list("Constraint Types"
        list("Alignment" ))
        list("Member Types" )
        )
```

ciCurrentPathIterator

```
ciCurrentPathIterator(
     d_cellview
     t_matchExpr
     )
```

Description

Finds the current path structure in Circuit Prospector. It is an iterator function that does not support hierarchy.

Arguments

d_cellview	The cellview containing the design instances to be iterated over.
t_matchExpr	The matched expression string to be used in the iteration. Variables `"device"` and `"powerDevice"` can be used in `matchExpr`.

Value Returned

None

Example

```
ciCurrentPathIterator(cv  "t")
```

ciDefaultParamToMatchFilter

```
ciDefaultParamToMatchFilter(
     d_insts
     [ t_simulator ]
     )
     => l_param2DList / nil
```

Description

The function for the default filter in the Match Subset Editor.

It is defined as:

```
procedure( ciDefaultParamToMatchFilter(inst simulator)
    prog((param2DList)
    if( ciIsDevice(inst "fet") then
        param2DList= ciGetMatchParam2DList(inst "fetParamListForMatching")
    )
    if( ciIsDevice(inst "resistor") then
        param2DList = ciGetMatchParam2DList(inst "resistorParamListForMatching")
    )
    if( ciIsDevice(inst "capacitor") then
        param2DList = ciGetMatchParam2DList(inst "capacitorParamListForMatching")
    )
    return(param2DList)
    )
)
```

With the list of parameter names per device type defined by:

```
ciMapParam("fetParamListForMatching" '("model" "w" "l" "m"))
ciMapParam("resistorParamListForMatching" '("model" "w" "l" "r" "m"))
ciMapParam("capacitorParamListForMatching" '("model" "w" "l" "c" "m"))
```

Note: The above implementation can be overridden. The argument simulator is not used in the above implementation but is provided for use while overriding.

Arguments

d_insts	Given instances.
t_simulator	Simulator to be used in override.

Value Returned

l_param2DList	A 2D list containing the parameter name-value pair for all of the filtered parameters.
nil	Command failed.

Example

```
ciDefaultParamToMatchFilter("MN5" "")
```

ciDeleteModgenTopologies

```
ciDeleteModgenTopologies(
    g_modgen
    )
    => t / nil
```

Description

Deletes the database topology objects associated with the specified modgen along with the constraints on those topology objects.

See also ciSetModgenTopology for an example of how to create database topology objects for a modgen.

Arguments

g_modgen	The modgen constraint.

Value Returned

`t`	Database topology was deleted successfully.
`nil`	Database topology was not deleted.

Example

```
mgGetTopologyFromModgen(modgen)
=> db:0x299dc85c

ciDeleteModgenTopologies(modgen)
mgGetTopologyFromModgen(modgen)
=> nil
```

ciDeleteRuleGroup

```
ciDeleteRuleGroup(
     g_constraintGroupPointer
     )
     => t / nil
```

Description

Deletes a constraint group.

See also: ciAddRuleGroup.

Arguments

g_constraintGroupPointer	The constraint group to be deleted.

Value Returned

`t`	Constraint group deleted successfully.
`nil`	Constraint group not deleted.

Example

```
ciDeleteRuleGroup(cg1)
```

ciDeleteUnreferencedObjects

```
ciDeleteUnreferencedObjects(
     g_constraintGroupPointer
     )
     => t / nil
```

Description

Deletes the unreferenced objects.

When a design contains unreferenced objects and you try to create contraints in it, the constraints might be out-of-context. In such a scenario, use this function before you start to create the constraints.

Arguments

u_cache	The cache ID from where the unreferenced objects have to be removed.

Value Returned

`t`	The unreferenced objects were deleted successfully.
`nil`	The unreferenced objects were not deleted.

Example

```
ciDeleteUnreferencedObjects(ciGetCellView(geGetEditCellView())
```

ciDesignLCV

```
ciDesignLCV(
     u_cache
     )
     => l_libCellViewNames
```

Description

Returns a list containing the library, cell, and view names of the specified constraint cache. The design library, cell, and view refers to the schematic or layout cellview. If the cache is associated with a configuration or physical configuration, it is the top cell specified by the configuration.

Arguments

u_cache	The constraint cache ID.

Value Returned

l_libCellViewNames	List comprising the library, cell and view names of the associated schematic or layout returned in the following format:
	`( t_libName t_cellName t_viewName )`
	For example:
	`( "overview" "block1" "schematic" )`

Example

Get the library, cell, and view names of the current constraint cache:

```
ciDesignLCV(ciGetCellView())
=> ("overview" "block1" "schematic")
```

or

```
ciDesignLCV(ciGetCellView())
=> ("overview" "block1" "layout")
```

ciEnableAutoConstraintNotes

```
ciEnableAutoConstraintNotes(
     g_enable
     )
     => t / nil
```

Description

Provides an API alternative that allows you to enable or disable auto constraint notes.

Note: The GUI equivalent is the *Auto Constraint Notes* option in the Editor Options form.

Arguments

g_enable	Boolean argument where `t` is used to enable auto constraint notes and `nil` is used to disable them.

Value Returned

`t`	Command successful.
`nil`	Command failed.

Example

```
ciEnableAutoConstraintNotes(t)
```

ciExpandMembers

```
ciExpandMembers(
     l_designObjectList
     [ ?compress g_compress ]
     )
     => l_designObjectList
```

Description

Expands any iterated or repeated member in the passed member list. For example, MN1<0:2> gets expanded to MN1<0>, MN1<1>, and MN1<2>.

In the case of net repetitions, such as <*3>inp, by default these get expanded N times as inp, inp, inp. However, if the optional compress argument is set to t, then <*3>inp will be expanded to just inp. The same applies to net bundles that contain repetitions, such as <*3>inp, a, b, c, inp, inp will by default be expanded to inp, inp, inp, a, b, c, inp, inp but with compress set to t will be expanded to inp, a, b, c.

Arguments

l_designObjectList	Specify the iterated or repeated member.
?compress *g_compress*	When set to t, any repetitions will only be expanded to a single bit rather than N bits.

Value Returned

l_designObjectList	Returns the expanded iterated or repeated member in the passed member list.

Example

```
instsNetsPins = list( list("MN1<4:0>" 'inst) list("MN0" 'inst) list("<*3>inp" 'net)
)
expandedMembers = ciExpandMembers(instsNetsPins ?compress t)

mapcar( lambda( (mem) car(mem) ) expandedMembers)
("MN1<4>" "MN1<3>" "MN1<2>" "MN1<1>" "MN1<0>" "MN0" "inp")
```

ciExpandName

```
ciExpandName(
     t_iteratedRepeatedBundledName
     g_compress
     s_type
     )
     => l_bitNames
```

Description

This function is used within constraint generators to expand iterated, repeated, and bundled names for instances, nets, and pins into a list of individual bit names. This is necessary if these names are to be used in calls to ciConCreate().

As an alternative to calling `ciExpandName()` and prior to calling ciConCreate(), call ciConCreateExpanded(), which does a default expansion of the iterated, repeated, and bundled names.

Arguments

l_iteratedRepeatedBundledName	Specify a string containing the iterated, repeated, and bundled name to be expanded.
g_compress	Specify to remove duplicate bit names from the list.
s_type	Specify the type of iterated object, that is, `'inst`, `'net`, or `'pin`.

Value Returned

l_bitNames	Returns a list of the individual bit names for the iterated, repeated, and bundled names.

Example

```
ciExpandName("<*2>(<*3>a,b),b,c<0:3>" nil 'inst)
("a" "a" "a" "b" "a" "a" "a" "b" "b" "c<0>" "c<1>" "c<2>" "c<3>")

ciExpandName("<*2>(<*3>a,b),b,c<0:3>" t 'net)
("a" "b" "c<0>" "c<1>" "c<2>" "c<3>")
```

```
ciExpandName("pin5<0:2>:3" nil 'pin)
("pin5<0>" "pin5<1>" "pin5<2>")

ciExpandName("pin5<0:2>" nil  'pin)
("pin5<0>" "pin5<1>" "pin5<2>")

ciExpandName("pin3<10>:pin3<10>" nil 'pin)
pin3<10>
```

ciFindOpenCellView

```
ciFindOpenCellView(
     t_libName
     t_cellName
     t_viewName
     t_constraintViewName
     )
     => u_cache / nil
```

Description

Returns the open constraint cache for the library, cell, view, and constraint view. If the constraint cache is not already open, this SKILL function returns `nil`.

See also ciCacheFind that finds the constraint cache based on the constraint `viewNameList`.

See also ciFindOpenCellViews that allows a wider range of criteria to be matched.

Arguments

t_libName	The library name.
t_cellName	The cell name.
t_viewName	The view name.
t_constraintViewName	The constraint view name.

Value Returned

u_cache	Returns the constraint `cache`.
nil	No open constraint cache could be found for the specified library, cell, view, and constraint view.

Example

- Find the open constraint cache associated with a `physConfig`.

```
cc = ciFindOpenCellView("overview" "block1" "physConfig" "constraint")
```

- Find the open constraint cache for an alternative constraint view:

```
constrviewname = "constr"
```

```
unless( cc = ciFindOpenCellView("overview" "block1" "schematic"
             constrviewname)
  info("%s is not open" constrviewname)
)
```

ciFindOpenCellViews

```
ciFindOpenCellViews(
     [ ?cacheLib t_cacheLib ]
     [ ?cacheCell t_cacheCell ]
     [ ?cacheView t_cacheView ]
     [ ?designLib t_designLib ]
     [ ?designCell t_designCell ]
     [ ?designView t_designView ]
     [ ?constraintLib t_constraintLib ]
     [ ?constraintCell t_constraintCell ]
     [ ?constraintView t_constraintView ]
     )
     => l_cacheIds / nil
```

Description

Returns a list of open constraint caches that match the criteria specified by the optional keyed arguments. A constraint cache can be associated with the following three views:

1. The cache library, cell, and view refers to the schematic, configuration, physical configuration, or layout view that the constraint cache is associated with.

2. The design library, cell, and view refers to the schematic or layout cellview. If the cache is associated with a configuration or physical configuration, it is the top cell specified by the configuration.

3. The constraint library, cell, and view refers to the storage location of the constraints, which can be a constraint view or layout.

Arguments

`?cacheLib` *t_cacheLib*	The cache library/cell/view name.
`?cacheCell` *t_cacheCell*	
`?cacheView` *t_cacheView*	
`?designLib` *t_designLib*	The design library/cell/view name.
`?designCell` *t_designCell*	
`?designView` *t_designView*	

`?constraintLib` *`t_constraintLib`*

`?constraintCell` *`t_constraintCell`*

`?constraintView` *`t_constraintView`*

The constraint library/cell/view name.

Value Returned

`l_cacheIds`	A list of cache IDs.
`nil`	No open constraint cache matching the specified criteria could be found.

Example

- Find constraint caches using a constraint view name, as shown below.

```
ciFindOpenCellViews(?constraintView "constraint")
=> (ci:0x2d87cef0 ci:0x2d03a850 ci:0x2d038a00 ci:0x2d036910 ci:0x2cf95bf0)
ciFindOpenCellViews(?constraintView "constrBlock")
=> (ci:0x2d86f840)
```

- Find constraint caches associated with a physConfig, as shown below.

```
ciFindOpenCellViews(?cacheView "physConfig")
=> (ci:0x2d86f840 ci:0x2cf95bf0)
```

- Find constraint caches associated with a schematic, as shown below.

```
ciFindOpenCellViews(?cacheView "schematic")
=> (ci:0x2d87cef0 ci:0x2d03a850 ci:0x2d038a00 ci:0x2d036910)
```

- Additional constraint caches might have schematic as the design view, including those associated with a physConfig, as shown below.

```
ciFindOpenCellViews(?designView "schematic")
=> (ci:0x2d87cef0 ci:0x2d86f840 ci:0x2d03a850 ci:0x2d038a00 ci:0x2d036910
    ci:0x2cf95bf0)
```

- Find constraint caches using multiple criteria, as shown below.

```
ciFindOpenCellViews(?cacheLib "overview" ?designView "layout")
=> (ci:0x26fc4820 ci:0x243626a0)
ciFindOpenCellViews(?cacheView "physConfig" ?designLib "overview"
?constraintView "constraint")
=> (ci:0x2701bf80)
```

- Layout constraint caches might be found using the cache, design and constraint names, as shown below.

```
ciFindOpenCellViews(?cacheView "layout")
=> (ci:0x2d6ef0c0 ci:0x2a374e80)
=> ciFindOpenCellViews(?designView "layout")
(ci:0x2d6ef0c0 ci:0x2a374e80)
ciFindOpenCellViews(?constraintView "layout")
=> (ci:0x2d6ef0c0 ci:0x2a374e80)
```

- Multiple constraint caches might be associated with a schematic, as shown below.

```
ciFindOpenCellViews(?designLib "overview" ?designCell "block1" ?designView
"schematic")
=> (ci:0x24304250 ci:0x24370340)
```

- Schematic is never the constraint storage view, as shown below.

```
ciFindOpenCellViews(?constraintView "schematic")
=> nil
```

- Constraint is never the primary cache view, as shown below.

```
ciFindOpenCellViews(?cacheView "constraint")
=> nil
```

ciGetCellTermDefaultNetName

```
ciGetCellTermDefaultNetName(
     t_cellName
     t_termName
     )
     => t_netName / nil
```

Description

Returns the default net name registered with a terminal for the cell name and terminal name that is passed.

Note: The default net is set using ciRegisterDefaultNetName.

Arguments

t_cellName	The cell name.
t_termName	The terminal name.

Value Returned

t_netName	The default net name registered with the given cell name and terminal name.
nil	No default net name registered.

Example

```
ciGetDefaultNetName("pmos" "D")
```

ciGetCellView

```
ciGetCellView(
     [ ?window d_window ]
     [ ?findOnly g_findOnly ]
     [ @rest rest ])
     => u_cache / nil
```

Description

Returns the constraint cellview associated with a schematic or layout window.

Note: The `ciGetCellView` function can accept either of the two positional arguments, or will accept a single keyword as `ciGetCellView(?window windowId ?findOnly t)`.

Arguments

`?window` *`d_window`*	Name of a schematic or layout window. Defaults to the current window. **Note:** Positional and keyed arguments are accepted.
`?findOnly` *`g_findOnly`*	Specifies that the constraint cache is returned only if it is already open. Defaults to `nil`.

`@rest` *`rest`*	This is a special SKILL argument type which means zero or more unnamed arguments. This is because `ciGetCellView()` can be called with or without keyed arguments. For example, these are all equivalent:
	`ciGetCellView(?window win ?findOnly t)`
	`ciGetCellView(?window win t)`
	`ciGetCellView(win ?findOnly t)`
	`ciGetCellView(win t)`

Value Returned

`u_cache`	The constraint cache ID for the returned constraint cellview.
`nil`	Failed to return constraint cache ID.

Example

```
ciGetCellView(window(4))
```

Opens the constraint cache for window(4).

```
ciGetCellView(?window window(3) t)
```

Finds the constraint cache for window(3), if it is open.

```
ciGetCellView(?findOnly t)
```

Find the constraint cache for current window, if it is open.

```
ciGetCellView()
```

Opens the constraint cache for current window.

ciGetCellViewForObjectPath

```
ciGetObjectCellViewObjectPath(
     t_objectFullPathName
     t_currentHierPath
     )
     => d_cellviewID / nil
```

Description

Retrieves the cellview ID from the specified path to an object. This function is used within constraint generators.

Note: This function does not check if the object specified at the end of the path exists in the returned cellview. If the characters in the object path are not separated by a forward slash (for example, `objectName` = `"/abc/"` or `""` or `"///"`, and so on), it is assumed that the object would be found in the current cellview. If the object is found, the following function returns the cellview ID:

```
ciGetCellViewForObjectPath(objectName ciGetCellView())
```

Arguments

t_objectFullPathName	Full hierarchical path to the object including the object name.
t_currentHierPath	Current path in the hierarchy.

Value Returned

d_cellviewID	Returns cellview ID associated with a particular object path.
`nil`	Failed to return a cellview ID.

Example

```
cv = ciGetCellViewForObjectPath("/I16/MP0" "")
```

```
cv->libName
"amsPLL"
cv->cellName
"vco2phase"
cv->viewName
"schematic"
```

ciGetConnectedInsts

```
ciGetConnectedInsts(
     d_cvID
     l_nets
     u_constraint
     )
     => l_instsNets / nil
```

Description

Finds the instances to which all the nets in a constraint are connected and the corresponding connecting nets on which the constraint can be propagated.

Arguments

d_cvID	Cellview ID to which the specified nets and constraint belong.
l_nets	List of nets belonging to the specified constraint.
u_constraint	The constraint containing the specified nets.

Value Returned

l_instsNets	Returns a list of lists. The first member of the list is a connected instance and second is a list of connecting nets. For example: `list(list(inst1 list(net1 net2)) list(inst2 list(net3 net4)) ...)`
`nil`	Failed to return the list.

Example

```
cv=geGetEditCellView()
db:0x230ae61a

con=ciConFind(ciGetCellView() "CG__0")
ci:0x2dee5dd0

nets = list("inp3" "inp4")
("inp3" "inp4")
```

```
insts = ciGetConnectedInsts(cv nets con)
    ((db:0x230a88e6
        ("outp" "outm")
    )
    (db:0x230a88e4
        ("inp" "inm")
    )
)

car(insts)
    (db:0x294888e6
        ("outp" "outm")
    )

caar(insts)
db:0x294888e6

caar(insts)~>name
"|I16"
```

ciGetCustomFilterNames

```
ciGetCustomFilterNames(
      )
      => l_filterList / nil
```

Description

Returns a list of registered custom filters (including the *Default* filter) for the Match Subset Editor.

Arguments

None

Value Returned

l_filterList	List of registered custom filters
nil	Failed to return constraint cache ID.

Example

If a custom filter has been registered as `myFilter`, then

```
ciGetCustomFilterNames()
```

will return

```
("Default" "myFilter")
```

ciGetDefaultNetName

```
ciGetDefaultNetName(
     d_inst
     t_termName
     )
     => t_netName / nil
```

Description

Returns the default net name registered with a terminal by specifying and an instance ID and a terminal name.

Arguments

d_inst	The instance ID.
t_termName	The terminal name.

Value Returned

t_netName	The default net name registered with the terminal.
nil	No registered name found.

Example

```
ciGetDefaultNetName(db:0x345678 "B")
```

ciGetFoundryRules

```
ciGetFoundryRules(
     d_techfile
     )
     => oaConstraintGroup / nil
```

Description

Returns the foundry constraint group for the given technology file.

Arguments

d_techfile	The technology file.

Value Returned

oaConstraintGroup	The `oaConstraintGroup` for the given technology.
`nil`	Command failed.

Example

```
cv=geGetEditCellView() tf=techGetTechFile(cv) ciGetFoundryRules(tf)~>name
```

ciGetMatchParam2DList

```
ciGetMatchParam2DList(
     d_insts
     t_holdername
     )
     => l_param2DList / nil
```

Description

Returns a 2D list (name-value pair) of CDF parameters present on the given instances as well as in the given holder list.

Arguments

d_insts	Instances where name-value pairs are required to be returned.
t_holdername	Holder list where name-value pairs are required to be returned.

Value Returned

l_param2DList	2D list (name-value mapping) of CDF parameters.
`nil`	Command failed.

Example

If there is an instance, `inst`, the following SKILL function:

```
ciGetMatchParam2DList(inst "fetParamListForMatching")
```

Displays:

```
(("model" "pmos1")
    ("l" "340.0n")
    ("w" "8u")
    ("m" "1")
)
```

ciGetMembersOfType

```
ciGetMembersOfType(
     l_instsNetsPins
     s_type
     [ ?includeType g_includeType ]
     [ ?removeLeadingSlash g_removeLeadingSlash ]
     [ ?expand g_expand ]
     )
     => l_filteredList
```

Description

Filters the `instsNetsPins` list used by constraint generators to only include members of a specified type. See also, ciConGetMembersOfType.

Arguments

l_instsNetsPins	A list of lists where each sub-list contains an instance, net, or pin name, and the appropriate type symbol ('`inst`, '`net`, '`pin`, '`instTerm`).
s_type	The member type by which the `instsNetsPins` list is to be filtered.
`?includeType` *g_includeType*	Controls whether the returned filtered list includes the type symbols or not. Default: `t`
`?removeLeadingSlash` *g_removeLeadingSlash*	Controls whether the leading slash (if present) should be removed from instance, net, pin, or instTerm names. Default: `t`
`?expand` *g_expand*	Controls whether the returned member names are expanded. Default: `nil`

Value Returned

l_filteredList	A filtered version of the passed `instsNetsPins` list, which only contains items of the specified type.

Example

```
instsNetsPins = '(("MN1" inst) ("MN2" inst) ("netA" net) ("pinA" pin) ("MN3" inst))
ciGetMembersOfType( instsNetsPins 'inst )
=> '(("MN1" inst) ("MN2" inst) ("MN3" inst))

ciGetMembersOfType( instsNetsPins 'inst ?includeType nil)
=> '("MN1" "MN2" "MN3")

instsNetsPins = '(("MN1<0:1>" inst) ("netA" net) ("pinA" pin))
ciGetMembersOfType( instsNetsPins 'inst ?expand t)
=> '(("MN1<0>" inst) ("MN1<1>" inst))
```

ciGetObjectCellView

```
ciGetObjectCellView(
     t_objectPathAndName
     l_cache
     )
     => cvID / nil
```

Description

Used within constraint generators to retrieve the cellview associated with a particular object path. This SKILL function is similar to `ciGetCellViewForObjectPath`, but uses the cellview information associated with the cache if the object path does not contain a hierarchical path.

Note: This function does not check if the object specified at the end of the path exists in the returned cellview. If the characters in the object path are not separated by a forward slash (for example, `objectName` = `"/abc/"` or `""` or `"///"`, and so on), it is assumed that the object would be found in the current cellview. If the object is found, the following function returns the cellview ID:

```
ciGetCellViewForObjectPath(objectName ciGetCellView())
```

Arguments

t_objectPathAndName	Specifies the cellview path and object name.
l_cache	Specifies the cache ID.

Value Returned

cvID	Returns cellview ID associated with a particular object path.
`nil`	The command failed.

Example

```
cv = ciGetObjectCellView("/I16/MP0" cacheTop)
```

```
cv->libName
"amsPLL"
cv->cellName
"vco2phase"
cv->viewName
"schematic"
```

```
;; Without a hier path to MP0 will use the cellview information associated with the
cache to locate the cell view for MP0

cv = ciGetObjectCellView("MP0" cache_vco2phase)

cv->libName
"amsPLL"
cv->cellName
"vco2phase"
cv->viewName
"schematic"


;; Without a hier path to MP0 will use the cellview information associated with the
cache to locate the cell view for MP0, but cacheTop does not contain MP0 - return nil

cv = ciGetObjectCellView("MP0" cacheTop)

cv->libName
nil
cv->cellName
nil
cv->viewName
nil
```

ciGetOpenCellViews

```
ciGetOpenCellViews(
     [ ?includeLayout g_includeLayout ]
     )
     => l_cacheIds / nil
```

Description

Returns a list of open constraint caches in the virtual memory.

Arguments

`?includeLayout` *`g_includeLayout`*

Specifies whether the layout constraint caches are included. Default: `nil`

Value Returned

`l_cacheIds`	A list of cache IDs.
`nil`	No open constraint cache matching the specified criteria could be found.

Example

- Get the modified constraint views:

```
modifiedcaches = setof(c ciGetOpenCellViews() ciCacheIsModified(c))
```

- Get the open constraint library/cell/view names:

```
mapcar('ciConstraintLCV ciGetOpenCellViews())
```

- Sort the list of constraint caches, excluding layout:

```
sort(ciGetOpenCellViews(?includeLayout nil) 'ciConstraintViewLessp)
```

- Get the layout constraint caches:

```
layoutcaches = setof(c ciGetOpenCellViews(?includeLayout t)
ciCacheIsLayout(c))
```

ciGetRuleGroupByName

```
ciGetRuleGroupByName(
     [ d_techfile | d_cellView ]
     t_name
     )
     => oaConstraintGroup / nil
```

Description

Returns the constraint group from the specified design or technology database with the specified name.

Arguments

d_techfile | *d_cellView*	Technology file or design cellview.
t_name	Name of cellview or techfile.

Value Returned

oaConstraintGroup	Constraint group.
nil	Command failed.

Example

```
ciGetRuleGroupByName(tf, "2xRouting")
```

ciGetRuleGroupName

```
ciGetRuleGroupName(
     g_constraintGroupPointer
     )
     => oaConstraintGroup / nil
```

Description

Returns the string name of a constraint group from a given constraint group pointer.

Arguments

g_constraintGroupPointer	The constraint group pointer for which string name is needed.

Value Returned

oaConstraintGroup	The name of the constraint group for the given constraint group pointer.
nil	Command failed.

Example

```
ciGetRuleGroupName(cg1)
"cg1"
```

ciGetRuleGroups

```
ciGetRuleGroups(
     [ d_techID | d_cellViewID ]
     )
     => oaConstraintGroupList / nil
```

Description

Returns a list of `oaConstraintGroup` pointers for a given technology or design file.

Arguments

d_techID	The technology file identifier.
d_cellViewID	The cellview identifier.

Value Returned

oaConstraintGroupList	List of `oaConstraintGroup` pointers.
`nil`	Command failed.

Example

```
ciGetRuleGroupName(cg1)
"cg1"
```

ciGetWidgetProperties

```
ciGetWidgetProperties(
     l_genArgs
     )
     => l_widgetProp / nil
```

Description

Returns the widget properties set on the constraint generator arguments, `genArgs`.

Arguments

l_genArgs	The list of arguments values (`args` or `oldArgs`) that are passed to an argument callback function of a constraint generator. For example: `genArgs = list(nil 'argName1 value1 ... 'argNameN valueN 'widgetProperties list('widgetProperties 'widgetName1 list('widgetPropertyList 'propName1 propName1Value ...) ...) )`

Value Returned

l_widgetProp	A disembodied property list containing the widget properties of the given arguments in the following format: `('widgetProperties 'widgetName1 list('widgetPropertyList 'propName1 propNameValue1 ...) ... 'widgetNameN list('widgetPropertyList 'propNameA propNameValueA ...) )` If `genArgs` does not match the format of the arguments list passed to the callback function of a constraint generator argument, the result is undefined.
`nil`	There are no associated widget properties.

Example

```
args = list(nil 'width 0.35 'length 0.7
    'widgetProperties list('widgetProperties
```

```
    'width list('toolTip "" 'hide nil 'enable t)
    'height list('toolTip "" 'hide nil 'enable t)
    'width\.label list('text "Width" 'toolTip "" 'hide nil 'enable t)
    'height\.label list('text "Height" 'toolTip "" 'hide nil 'enable t)
    )

procedure(myArgumentCallback(cache argName args oldArgs instsNetsPins userEdit)
let(( (properties ciGetWidgetProperties(args) ) )
;;Setting a toolTip on the widget width
properties->width->toolTip = "Enter the width of the transistor"
;;changing the text displayed in the constraint generator dialog for the argument
widget
properties->width\.label->text = "Width (um):"

;;returning the updated value of argument
;; args is now equal to:
;; list(nil 'width 0.35 'length 0.7
;;  'widgetProperties list('widgetProperties
;;  'width list('toolTip "Enter the width of the transistor" 'hide nil 'enable t)
;;  'height list('toolTip "" 'hide nil 'enable t)
;;  'width\.label list('text "Width (um):" 'toolTip "" 'hide nil 'enable t)
;;  'height\.label list('text "Height" 'toolTip "" 'hide nil 'enable t)
;;  )

args

)
)
```

ciHasCellAnyRegTerm

```
ciHasCellAnyRegTerm(
     t_cellName
     )
     => t / nil
```

Description

Returns information on whether or not the given cell has at least one terminal registered with a default net.

Arguments

t_cellName	The cell name.

Value Returned

t	Given cell has one or more terminals registered with a default net.
nil	No terminals registered.

Example

```
ciHasCellAnyRegTerm("pmos")
```

ciHaveSameBulkNets

```
ciHaveSameBulkNets(
     l_dbId
     )
     => t / nil
```

Description

Returns `t` if the passed list of devices have the same bulk connection.

This function relies on the bulk terminal names that are being registered for the devices in the PDK using `ciMapTerm("bulk" <listOfBulkTerminalNames>)`.

Arguments

l_dbId	Specifies a list of devices.

Value Returned

`t`	If the specified list of devices have the same bulk connection.
`nil`	No same bulk connection.

Example

```
l_dbId = geGetSelectedSet()
ciHaveSameBulkNets(l_dbId)
```

ciHierCompareConstraint

```
ciHierCompareConstraint(
     u_sourceCons
     u_targetCache
     [ ?paths l_occurrencePaths ]
     [ ?recursive g_recursive ]
     )
     => constraintComparisons / nil
```

Description

Copies and compares applicable pushed or pulled constraints in the layout hierarchy. The copied constraint members, such as nets, terminals, instance terminals, and so on correspond to the original constraint members as they are physically connected.

Arguments

u_sourceCons	The source cache in the lower-level layout.
u_targetCache	The destination constraint cache in the upper-level design hierarchy.
`?paths` *l_occurrencePaths*	The list of paths where constraints are to be compared. If not set, constraints are compared in all occurence paths. Default: `nil`
`?recursive` *g_recursive*	When set to `t`, constraints are compared on as many levels as possible in all applicable occurrence paths. Default: `nil`

Value Returned

constraintComparisons	Details of pushed or pulled constraint comparisons.
`nil`	No constraint comparison made.

ciHierCompareConstraints

```
ciHierCompareConstraints(
     u_sourceCache
     u_targetCache
     [ ?paths l_occurrencePaths ]
     [ ?recursive g_recursive ]
     )
     => constraintComparisons / nil
```

Description

Compares applicable pushed or pulled constraints in the layout hierarchy. The copied constraint members (nets, terminals, instance terminals, and so on) correspond to the original constraint members because they are physically connected.

Arguments

u_sourceCache	The source cache in the upper-level layout
u_targetCache	The destination constraint cache down the design hierarchy
?paths *l_occurrencePaths*	The list of paths where constraints are to be compared. If not set, constraints are compared in all occurence paths. Default: `nil`
?recursive *g_recursive*	When set to `t`, constraints are compared on as many levels as possible in all applicable occurrence paths. Default: `nil`

Value Returned

constraintComparisons	Details of pushed or pulled constraint comparisons.
`nil`	No constraint comparison made.

Examples

```
ciHierCompareConstraints(source target)
ciHierCompareConstraints(source target ?paths list("/I0" "/I1") ?recursive t)
ciHierCompareConstraints(source target ?paths list("/I3/I0/I0"))
ciHierCompareConstraints(source target ?recursive t)
```

ciHierUpdateConstraints

```
ciHierUpdateConstraints(
     u_sourceCache
     u_targetCache
     [ ?paths l_occurrencePaths ]
     [ ?recursive g_recursive ]
     )
     => t / nil
```

Description

Used in sync with the pushed and pulled constraints with their source constraints. After push or pull the constraint parameters have been modified, you can use this SKILL function to update the pushed or pulled constraints.

Arguments

u_sourceCache	Cache for source constraints.
u_targetCache	Cache where constraints have been created after push or pull. Used only for pulled constraints.
?paths *l_occurrencePaths*	Update constraints for specific instances.
?recursive *g_recursive*	Update constraints recursively, if `true`.

Value Returned

`t`	If any constraint gets updated.
`nil`	If no constraint is updated.

Example

```
ciHierUpdateConstraints(
     srcCache destCache ?recursive t
     )
```

ciIsNetSuperType

```
ciIsNetSuperType(
     t_netType
     )
     => t / nil
```

Description

Tests whether or not the given net type is a super-type.

Arguments

t_netType	A net type.

Value Returned

t	If the net type is a super-type.
nil	If the net type is not a super-type.

Example

```
;; Register a super type and some sub-types.
ciRegisterNetSuperType("Supply" '("Power" "Ground"))
ciRegisterNetNames("Power" '("vcc"))
ciRegisterNet("Ground" '("gnd"))

;; Check which types are super and which are sub.
ciIsNetSuperType("Supply") ; t
```

The above example shows that `Supply` is a super-type.

```
ciIsNetSuperType("Power") ; nil
```

The above example shows that `Power` is not a super-type.

```
ciIsNetSuperType("Ground") ; nil
```

The above example shows that `Ground` is not a super-type.

ciLoadConfigXML

```
ciLoadConfigXML(
     t_directoryPathToConfigXML
     )
     => g_result
```

Description

Loads the specified constraint `config.xml` file. In the process, the new entries get merged into the existing `config.xml` files that have already been loaded and the existing entries are overwritten.

When you start Virtuoso, all `config.xml` files found in the Cadence Search Path are loaded and merged based on the order in which they are found in the `setup.loc` file. By using this function, you can load a `config.xml` on demand after Virtuoso has been started, such as when a design library or PDK is loaded.

See also ciLoadDotCadenceFiles, ciLoadIcon, and ciLoadIcons.

Arguments

t_directoryPathToConfigXML	The full path to the directory that contains a file named `config.xml`.

Value Returned

g_result	Boolean value indicating success or failure.

Example

```
ciLoadConfigXML("/project1/.cadence/dfII/ci") ;; load project specific config.xml
ciLoadConfigXML("./.cadence/dfII/ci") ;; load local config.xml
```

ciLoadConfigXMLFromString

```
ciLoadConfigXMLFromString(
     t_configXMLString
     )
     => g_result
```

Description

Loads the string representation of a `config.xml` file. If a new entry does not match an existing entry, the new entry gets added into the `config.xml` files that have already been loaded. However, if the new entry matches an existing entry, the new entry overwrites the old one.

When you start Virtuoso, all `config.xml` files found in the Cadence Search Path are loaded and merged based on the order in which they are found in the `setup.loc` file. By using this function, you can load a `config.xml` on demand after Virtuoso has been started, such as when a design library or PDK is loaded.

See also ciLoadConfigXML, ciLoadDotCadenceFiles, ciLoadIcon, and ciLoadIcons.

Arguments

t_configXMLString	The string representation of a `config.xml` file.

Value Returned

g_result	Boolean value indicating success or failure.

Example

```
configXMLstring = "<ConstraintConfig>
  <ConstraintType>
    <Name>myConType</Name>
    <GUIName>My Con Type</GUIName>
    <Icon>myConTypeIcon.png</Icon>
    <MinMembers>1</MinMembers>
    <MaxMembers>1000000</MaxMembers>
    <AllowedMemberTypes>inst</AllowedMemberTypes>
    < VerifyCB>myConTypeVerifyCB</VerifyCB>
    <Param>
```

```
        <Name>param1</Name>
        <GUIName>Param1</GUIName>
        <Type>boolean</Type>
        <Scope>constraint</Scope>
        <DefaultValue>true</DefaultValue>
    </Param>
    <Param>
        <Name>Param2</Name>
        <Type>enum</Type>
        <ValueRange>aaa bbb ccc ddd</ValueRange>
        <Scope>constraint</Scope>
        <DefaultValue>\"ccc\"</DefaultValue>
    </Param>
  </ConstraintType>
</ConstraintConfig>"
```

ciLoadConstrFrom

```
ciLoadConstrFrom(
     t_sourceCache
     t_targetCache
     t_mode
     )
     => t / nil
```

Description

Loads the constraints from the source cache to the target cache of the same type.

Arguments

t_sourceCache	The source cache.
t_targetCache	The target cache.
t_mode	The mode in which the constraints should be loaded from the source cache. The following values are accepted: ■ `"Replace"` ■ `"Append"`

Value Returned

`t`	Returns `t` if the constraints were successfully loaded from the specified source cache to the target cache. In addition, an INFO message is displayed in the CIW listing the count of constraints, parasitic constraints, and constraint groups that were successfully loaded from a specific cellview.
`nil`	Returns `nil` if the constraints could not be loaded from the specified source cache to the target cache.

Example

```
ciLoadConstrFrom(ciOpenCellView("amsPLL" "vco2phase" "schematic"
"constraint_source" "r") ciOpenCellView("amsPLL" "vco2phase" "schematic"
"constraint_target" "r") "Replace")

=> t

INFO (CMGR-5294): Loaded the following successfully from cellview 'amsPLL/
vco2phase/schematic':

    3 of 3 constraints

    2 of 4 parasitic constraints

    1 of 2 constraint groups
```

ciLoadDotCadenceFiles

```
ciLoadDotCadenceFiles(
     t_pathToDotCadenceDir
     [ g_verbose ]
     )
     => g_result
```

Description

Loads all constraint-related files within the hierarchy of the specified `.cadence` directory.

Specifically the following type of files are loaded:

- `.cadence/icons/16x16/*.png`
- `.cadence/dfII/ci/config.xml`
- `.cadence/dfII/ci/iterators/*.il *.ile *.cxt`
- `.cadence/dfII/ci/generators/*.il *.ile *.cxt`
- `.cadence/dfII/ci/structures/*.il *.ile *.cxt`
- `.cadence/dfII/ci/finders/*.il *.ile *.cxt`
- `.cadence/dfII/ci/categories/*.il *.ile *.cxt`
- `.cadence/dfII/ci/categories/org/*.il *.ile *.cxt`

In case of the `config.xml` file, the new entries get merged into the existing `config.xml` files that have already been loaded and the existing entries are overwritten.

This SKILL function can be called within the `libInit.il` file of a PDK or design library so that library-specific constraint customizations can be setup when the library is loaded. As you can have your own customizations, it is recommended that the following command is run to re-load those customizations:

```
ciLoadDotCadenceFiles("./.cadence")
```

Arguments

t_pathToDotCadenceDir	The full path to the `.cadence` directory containing the files to be loaded.
g_verbose	(Optional) Boolean to turn on verbose messages about the files being loaded. It defaults to `nil`.

Value Returned

g_result	Returns `t` or `nil` depending on whether the files were read correctly.

Example

```
ciLoadDotCadenceFiles("./.cadence") ;;; load constraints files from local .cadence
directory
```

```
ciLoadDotCadenceFiles(getShellEnvVar("PROJECT_DOT_CADENCE") t) ;;; load
constraints files from a project specific .cadence directory and turn on verbose
messages:
```

```
INFO (CMGR-3066): Loading all files within '/project1/skill/.cadence'...
INFO (CMGR-3066): Loading icons from '/project1/skill/.cadence/icons/16x16'...
INFO (CMGR-3066): Loading icon file: '/project1/skill/.cadence/icons/16x16/
MyCascode.png'...
INFO (CMGR-3066): Loading icon file: '/project1/skill/.cadence/icons/16x16/
MyDifferentialPair.png'...
INFO (CMGR-3066): Loading XML config from '/project1/skill/.cadence/dfII/ci'...
INFO (CMGR-3066): Loading Constraint Editor files[0]...
INFO (CMGR-3066): Loading Constraint Generator files[2]...
INFO (CMGR-3066):    /project1/skill/.cadence/dfII/ci/generators/MyCascode.il...
INFO (CMGR-3066):    /project1/skill/.cadence/dfII/ci/generators/
MyDifferentialPair.il...
INFO (CMGR-3066): Loading Circuit Prospector Iterator files[3]...
INFO (CMGR-3066):    /project1/skill/.cadence/dfII/ci/iterators/
MyCascodeIterator.il...
INFO (CMGR-3066):    /project1/skill/.cadence/dfII/ci/iterators/
MyDifferentialPairIterator.il...
INFO (CMGR-3066):    /project1/skill/.cadence/dfII/ci/iterators/
MyIteratorUtils.il...
INFO (CMGR-3066): Loading Circuit Prospector Finder files[2]...
INFO (CMGR-3066):    /project1/skill/.cadence/dfII/ci/finders/MyCascode.il...
INFO (CMGR-3066):    /project1/skill/.cadence/dfII/ci/finders/
MyDifferentialPair.il...
INFO (CMGR-3066): Loading Circuit Prospector Structure files[0]...
INFO (CMGR-3066): Loading Circuit Prospector Category files[1]...
INFO (CMGR-3066):    /project1/skill/.cadence/dfII/ci/categories/MyCategory.il...
INFO (CMGR-3066): Loading Circuit Prospector Category Organization files[0]...
INFO (CMGR-3066): Loading Constraint Filter files[0]...
INFO (CMGR-3066): Loading Constraint Menu History files[0]...
t
```

ciLoadIcon

```
ciLoadIcon(
     t_iconFilePath
     [ g_verbose ]
     )
     => g_result
```

Description

Loads a specific PNG format icon file from the specified directory so that it can be used by the Constraint Manager. The file should have a `.png` extension.

When you start Virtuoso, all icon files found in the `.cadence/icons/16x16` directories in the Cadence Search Path are loaded in the order in which they are found in the `setup.loc` file. By using this function, you can load icon files on demand after Virtuoso has been started, such as when a design library or PDK is loaded.

See also ciLoadDotCadenceFiles, ciLoadConfigXML, and ciLoadIcons.

Arguments

t_iconFilePath	The full path of the `.png` icon file to be loaded.
g_verbose	Boolean to turn on verbose messages about the icon files being loaded. It defaults to `nil`.

Value Returned

g_result	Returns `t` or `nil` indicating success or failure.

Example

```
ciLoadIcon("/project/.cadence/icons/16x16/Cascode.png") ;;; load project specific
icons
t
ciLoadIcon("./.cadence/icons/16x16/MyCascode.png" t) ;;; load local icons
verbosely
INFO (CMGR-3066): Loading icon file: './.cadence/icons/16x16/MyCascode.png'...
t
```

ciLoadIcons

```
ciLoadIcons(
     t_iconDirectoryPath
     [ g_verbose ]
     )
     => g_result
```

Description

Loads all PNG format icon files from the specified directory so that these can be used by the Constraint Manager. The file should have a `.png` extension.

When you start Virtuoso, all icon files found in the `.cadence/icons/16x16` directories in the Cadence Search Path are loaded in the order in which they are found in the `setup.loc` file. By using this function, you can load icon files on demand after Virtuoso has been started, such as when a design library or PDK is loaded.

See also ciLoadDotCadenceFiles, ciLoadConfigXML, and ciLoadIcon.

Arguments

t_iconDirectoryPath	The directory path of the `.png` icon files to be loaded.
g_verbose	(Optional) Boolean to turn on verbose messages about each icon file being loaded. It defaults to `nil`.

Value Returned

g_result	Returns `t` or `nil` indicating success or failure.

Example

```
ciLoadIcon("/project/.cadence/icons/16x16/Cascode.png") ;;; load project specific
ciLoadIcons("/project/.cadence/icons/16x16") ;;; load project specific icons
t
ciLoadIcons("./.cadence/icons/16x16" t) ;;; load local icons verbosely
INFO (CMGR-3066): Loading icon file: './.cadence/icons/16x16/MyCascode.png'...
INFO (CMGR-3066): Loading icon file: './.cadence/icons/16x16/
MyDifferentialPair.png'...
t
```

ciListEditors

```
ciListEditors(
     )
     => l_editors / nil
```

Description

Generates a list of the current constraint editors that are available from the *Constraint Manager's Constraint Generator* toolbar option.

Arguments

None

Value Returned

l_editors	List of current constraint editors.
nil	No constraint editors currently available.

ciListTypes

```
ciListTypes(
      )
      => l_types
```

Description

Generates a list of the current definitions for each constraint type in the `config.xml` files, the name of the constraint, and a list of legal constraint parameters and their default values.

Arguments

None

Value Returned

l_types	List of constraint type, name and parameters.

ciListProcessRules

```
ciListProcessRules(
     d_Id
     [ t_parameterType ]
     )
     => process_rules / nil
```

Description

Lists all process rules associated with the given object.

Arguments

d_Id	The database identifying number for the route or net. **Note:** The `ciListProcessRules` function also accepts cellviews.
t_parameterType	Optional string argument. Depending on what the object type is, the valid values for this argument are: ■ Default - for nets and constraints that have a *Default* param (this is the default value) ■ InputTaper - for nets ■ OutputTaper - for nets ■ Reflexive - for constraints that have a *Within Group* parameter ■ TransReflexive - for constraints that have a *Group to Outside Group* parameter ■ Interchild - for nested net classes (constraints that have a *Group to Group* parameter) ■ Shielding - for shielding constraints (constraints that have a *Shielding* parameter)

Value Returned

process_rules	A list showing the rules associated with the given object.
`nil`	Command failed.

Example

To list the input taper rules on a net:

```
ciListProcessRules(net, "InputTaper")
```

ciLxComparisonReport

```
ciLxComparisonReport(
     [ ?layoutCV g_layoutCV ]
     [ ?showReport g_showReport ]
     [ ?useViewNames g_useViewNames ]
     [ ?useTimeStamp g_useTimeStamp ]
     [ ?path g_path ]
     [ ?filename g_filename ]
     )
     => t / nil
```

Description

Prints a comparison report between a layout and its corresponding VLS XL schematic and/or top configuration.

Note: The layout should already be open in a VLS XL window.

Arguments

`?layoutCV` *`g_layoutCV`*	The layout cellview ID to be passed. Default: `nil`
`?showReport` *`g_showReport`*	Determines whether a browser with the report will be shown or not. Default: `t`
`?useViewNames` *`g_useViewNames`*	The file name created will contain the source and destination lib/cell/view names. This way the report files can be identified from the name without opening them. Default: `nil`
`?useTimeStamp` *`g_useTimeStamp`*	Determines whether or not a time stamp be used. If it is to be, the file name will contain a timestamp (year, month, day and time of day). Default: `nil`
`?path` *`g_path`*	Determines the path to be written to. Default: `nil`
`?filename` *`g_filename`*	Determines the file name to be written to. Default: `nil` **Note:** The file name will be auto-generated if either of the `useViewNames` or `useTimeStamp` argument is set. To generate a file with a user-defined name, do not specify any of these arguments.

Value Returned

`t`	Comparison report successfully generated.
`nil`	Comparison report not generated.

Example

- `ciLxComparisonReport ?layoutCV layoutCV_d`

  When passing in the layout cellview ID, the correct schematic or configuration will be identified and, when the design is already opened in VLS XL, a report will be generated for that layout and its current schematic pair.

- `ciLxComparisonReport ?showReport nil`

  A browser with the report will not be shown.

- `ciLxComparisonReport ?useViewNames t`

  The file name created will contain the source and the destination lib/cell/view names. This way the report files can be identified from the name without opening them.

- `ciLxComparisonReport ?useTimeStamp t`

  The file name contains a timestamp (year, month, day and time of day). It can be used in conjunction with any of the above examples.

  For example, if you want to generate a specific report for a layout cellview and its schematic, without showing a browser, with the file name identifying both the view names and the run time, then you can use...

- `ciLxComparisonReport ?layoutCV layoutCV_d ?showReport nil ?useViewNames t ?useTimeStamp t`

  The order in which the keyed parameters are placed does not matter.

- `ciLxComparisonReport ?path "pathstring/"`

This will write to the current directory that the path need not end with a `'/'` character. For example, `?path "/foo/bar"` will write a file in the `"/foo/bar/"`. The default is "./" and a string is expected.

- `ciLxComparisonReport ?path "temp" ?fileName "rpt.html"`

  This will create a file `rpt.html` under the `temp` directory.

ciModgenMergeLayersFromArgs

```
ciModgenMergeLayersFromArgs(
     l_args
     [ ?layersPresetArgName t_layersPresetArgName ]
     [ ?mergeLayersArgName t_mergeLayersArgName ]
     )
     => t_modgenLayers / null_string
```

Description

Returns a string with the list of layers from a property list, where each layer is separated by a comma.

Arguments

l_args — A property list. For example,

```
list(nil
Merge\ layers "layer1; layer2; layer3"
Layers\ preset presetValue
)
```

?layersPresetArgName *t_layersPresetArgName*

The name of the property containing the value for preset layers.

?mergeLayersArgName *t_mergeLayersArgName*

The name of the property containing the value for merge layers.

Value Returned

t_modgenLayers	A string of layers separated by commas.
null_string	If there are no layers, an empty string is returned.

Examples

- Assume, the default layers are: `layerDef1`, `layerDef2`, `layerDef3` and

```
args = list(nil Merge\ layers "layer1; layer2; layer3"
                Layers\ preset "default")
```

The function returns: `"layerDef1, layerDef2, layerDef3"`

- Assume, the well layers are: `layerWell1, layerWell2, layerWell3` and

  ```
  args = list(nil Merge\ layers "layer1; layer2; layer3 "  Layers\ preset "well")
  ```

  The function returns: `"layerWell1, layerWell2, layerWell3"`

- If there are no layers, but

  ```
  args = list(nil Merge\ layers "layer1; layer2; layer3 "  Layers\ preset "none")
  ```

  The function returns: `""`

- Assume,

  ```
  args =  list(nil Merge\ layers "layer1; layer2; layer3 "
                  Layers\ preset presetValue)
  ```

  where `presetValue` is a string different from `"default"`, `"none"`, or `"well"`.

  The function returns: `"layer1, layer2, layer3"`

ciModgenListFingerSplitCons

```
ciModgenListFingerSplitCons(
     g_modgen
     )
     => l_fingerSplitCons / nil
```

Description

Returns a list of `fingerSplit` constraints that exist on the passed modgen members. This SKILL function can be used in a template `create` or `check` callback function to include the `fingerSplit` constraints within a template.
Note: The `fingerSplit` constraints are not visible in the Constraint Manager assistant.

See also ciModgenSplitFingers and ciTemplateCreateDefinition.

Arguments

g_modgen	The modgen for which the `fingerSplit` constraints should be listed.

Value Returned

l_fingerSplitCons	The list of the `fingerSplit` constraints that are on the modgen members.
`nil`	`fingerSplit` constraints do not exist for the modgen members.

Examples

```
;;; Split the modgen fingers
ciModgenSplitFingers(ciGetCellView() modgen `(("MN1" inst) `(("MN2" inst)) t)
ciModgenListFingerSplitCons(modgen)~>name
    ("fs1" "fs2")
;;; unsplit the modgen fingers
ciModgenSplitFingers(ciGetCellView() modgen `(("MN1" inst) `(("MN2" inst)) nil)
ciModgenListFingerSplitCons(modgen)~>name
    nil
```

ciModgenRefreshStorage

```
ciModgenRefreshStorage(
     g_modgen
     )
     => t / nil
```

Description

Reinitializes the internal modgen storage. When the `ci` SKILL functions are used to update a modgen constraint in layout or the modgen previewer, the internal modgen storage needs to be reinitialized to ensure it is in a consistent state. Modgen constraint updates include adding/removing members, updating parameters, and adding/removing guard rings.

Arguments

g_modgen	The modgen that has been updated and whose storage needs to be reinitialized.

Value Returned

`t`	The modgen constraint was refreshed in the layout or the modgen previewer.
`nil`	The modgen constraint could not be refreshed in the layout or the modgen previewer.

Example

```
ciConUpdateParams(modgen '(("numRows" 4)))
ciModgenRefreshStorage(modgen)
```

ciModgenSplitFingers

```
ciModgenSplitFingers(
     g_cache
     g_modgen
     l_instsNetsPins
     g_split
     )
     => l_instsNetsPins
```

Description

Splits or unsplits the device fingers for the instances specified in the `instsNetsPins` list associated with the passed modgen. When called in a layout view, the physical device instances are split or unsplit and the modgen figGroups are updated accordingly. When called in a schematic view, the finger split information is stored in the schematic constraints cache. This information is used while transferring the schematic constraints to layout, and splitting or unsplitting the instances in layout.

This function should be called before the modgen members are updated to take into account device finger splitting. This function does not update the modgen members. This is the responsibility of the user because the interdigitation pattern of the modgen needs to accommodate the addition or removal of the finger-split instances.

See also ciModgenTemplateFingerSplitPreDestroy.

Arguments

g_cache	The constraints cache.
g_modgen	The modgen whose instances are being split or unsplit.
l_instsNetsPins	The devices to be split or unsplit. It is a list in the following format: `list((<devName1> 'inst) ... (<devNameN> 'inst))`
g_split	A Boolean controlling whether the fingers of the specified instances should be split or unsplit.

Value Returned

l_instsNetsPins	An updated `instsNetsPins` list containing the instance information after finger splitting or unsplitting.

Examples

```
instsNetsPins  = '(("|MN1" inst)("|MN2" inst))
               ;;; |MN1(m=1, fingers=2) |M2(m=1, fingers = 3)

splitInstsNetsPins = ciModgenSplitFingers(cache modgen instsNetsPins t)
            '(("|MN1.1" inst)("|MN1.2" inst)("|MN2.1" inst)("|MN2.2" inst)
              ("|MN2.3" inst))

ciModgenSplitFingers(cache modgen splitInstsNetsPins nil)
                '(("|MN1" inst)("|MN2" inst))
```

ciModgenTemplateFingerSplitPreDestroy

```
ciModgenTemplateFingerSplitPreDestroy(
     g_template
     )
     => t / nil
```

Description

Ensures that the split device instance fingers are unsplit and that the modgen figGroup is updated before the specified template is deleted. This high-level function should be used where modgens are contained within templates that control finger splitting on those modgens. To call this function before the template is deleted, it should be registered as the `preDestroySymbolName` of the template when the template is defined by using `ciTemplateCreateDefinition`.

See also ciModgenSplitFingers and ciTemplateCreateDefinition.

Arguments

g_template	The template that is about to be deleted.

Value Returned

`t`	Template deleted successfully after unsplitting the split device instance fingers and updating the modgen figGroup.
`nil`	Failed to delete the specified template.

Examples

```
ciTemplateCreateDefinition('FingerSplitTest
                            'FingerSplitTestCheckCB
                            'FingerSplitTestCreateCB
                            ?acceptsUserParams t
                            ?params ciGetStructArgs('FingerSplitTest)
                            ?preDestroySymbolName
                                ciModgenTemplateFingerSplitPreDestroy
                           )
```

ciObjectIsInContext

```
ciObjectIsInContext(
     u_cache
     t_designObjectName
     t_designObjectType
     )
     => t / nil
```

Description

Returns the in/out of context status of an object (Boolean value).

Arguments

u_cache	See cache.
t_designObjectName	See design_object_name.
t_designObjectType	See design_object_type.

Value Returned

t	Context status successfully returned.
nil	Context status not returned.

ciObjectListCon

```
ciObjectListCon(
     u_cache
     t_designObjectName
     t_designObjectType
     )
     => constraintIdList / nil
```

Description

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

Arguments

u_cache	See cache.
t_designObjectName	See design_object_name.
t_designObjectType	See design_object_type.

Value Returned

constraintIdList	A list of u_constraint for all objects that refer to the named design object as a member.
nil	No constraints returned.

ciObjectPathAndName

```
ciObjectPathAndName(
     t_objectPathAndName
     )
     => l_disembodied_objectPath_objectName
```

Description

This function is used in constraint generators to separate the object path and name from a string. The result is returned as a disembodied property list with properties, such as `objectPath` and `objectName`. The `objectPath` property is a list of strings that represent the full hierarchical path to the object including the objects name.

Arguments

t_ObjectPathAndName	The cellview path and object name.

Value Returned

l_disembodied_objectPath_objectName

Returns the disembodied property list.

Example

```
opn = ciObjectPathAndName("/I1/I2/MN1")


opn->objectPath ("I1" "I2" "MN1")
opn->objectName "MN1"
```

ciOpenCellView

```
ciOpenCellView(
     t_libName
     t_cellName
     t_viewName
     t_constraintViewName
     t_mode
     )
     => u_cache / nil
```

Description

Opens the constraint cache for the library, cell, view, and constraint view in read or edit mode as specified, and returns the constraint cache ID or returns `nil` if the constraint cache could not be opened.

`ciOpenCellView` allows the mode to be specified. However, it does not downgrade the mode to read if the constraint cache is already open in edit mode. In this case, `ciReopenCellView` can be used to change the mode, if required.

See also ciCacheGet that finds or opens the constraint cache based on the current constraint `viewNameList` instead of specifying a particular constraint view.

Arguments

t_libName	The library from which the constraint cache needs to be opened.
t_cellName	The cell from which the constraint cache needs to be opened.
t_viewName	The schematic view from which the constraint cache needs to be opened.
t_constraintViewName	The constraint view from which the constraint cache needs to be opened.
t_mode	The mode in which the constraint cache should be opened. The following values are accepted: ■ `"r"` for read mode ■ `"a"` for edit mode

Value Returned

u_cache	Returns the constraint cache.
nil	Returned when the specified library, cell, view, or constraint view does not exist.

Examples

- Open the constraint cache associated with a schematic in read mode.

```
cc = ciOpenCellView("overview" "block1" "schematic" "constraint" "r")
```

- Open the constraint cache associated with a physConfig in edit mode.

```
cc = ciOpenCellView("overview" "block1" "physConfig" "constrTampa" "a")
```

- Open the constraint cache for multiple constraint views in read mode.

```
constrviewnames = list("comstr1" "constr2" "constr3")
mapcar(
    lambda( (conview)
         ciOpenCellView("overview" "block2" "schematic" conview "r") )
    constrviewnames)
```

ciOpenPanicCellView

```
ciOpenPanicCellView(
     t_libName
     t_cellName
     t_viewName
     t_constraintViewName
     )
     => u_cache / nil
```

Description

Restores the constraint cellview (associated with a schematic cellview) that was saved in panic state, which is when Virtuoso exited unexpectedly, in the `hierDesign.oa-` file. The restored constraint cellview opens in append mode. You can use the Constraint Manager to view this constraint cellview. This SKILL function returns a cache pointer of the restored constraint cellview. Use this cache pointer as an argument with the `ciCacheSave` SKILL function to save the restored constraint cellview. You can then open the recovered constraint cellview in the Constraint Manager.

Arguments

t_libName	The library from which the constraint view needs to be restored.
t_cellName	The cell from which the constraint view needs to be restored.
t_viewName	The schematic view from which the constraint view needs to be restored.
t_constraintViewName	The constraint view that needs to be restored.

Value Returned

u_cache	Returns the constraint `cache`.
`nil`	Returned when the specified library, cell, view, or constraint view, or the `hierDesign.oa-` file does not exists.

Example

```
cache = ciOpenPanicCellView("amsPLL" "vco" "schematic" "constraint")
```

Here, `ciOpenPanicCellView` will open the cellview, `constraint` that was saved in panic state. Calling the `ciCacheSave(`*`cache`*`)` SKILL function will save the restored panic state view.

ciPrintReport

```
ciPrintReport(
     d_sourceCache
     d_targetCache
     [ t_fileName ]
     [ g_includeDefaultParam ]
     )
     => t / nil
```

Description

Generates an HTML report containing constraint differences between two designs.

The report contains the following sections:

- constraints only in source design
- constraints only in target design
- different constraints
- equal constraints

Arguments

d_sourceCache	The constraint `cache` for the source design (schematic).
d_targetCache	The constraint cache for the destination design (layout).
t_fileName	Optional argument which lets you set the html file name where the report should be written.
g_includeDefaultParam	Optional argument (default `t`) which lets you disable reporting default parameter values. Setting this parameter to `nil` generates a report that does not include any default constraint parameters.

Value Returned

`t`	Report successfully generated.
`nil`	Report failed to be created.

ciPullConstraint

```
ciPullConstraint(
     u_cons
     u_targetCache
     [ ?paths l_paths ]
     [ ?autoTag g_autoTag ]
     [ ?recursive g_recursive ]
     [ ?update g_update ]
     )
     => constraintIdList / nil
```

Description

Copies a single constraint up the layout hierarchy.

This can be described as "pulling" the constraint up the design hierarchy. The copied constraint's members (nets, terminals instance terminals and so on) will also correspond to the original constraint members as they are physically connected.

- Constraints will only be **pulled** if the `hierarchicalScope` parameter enumerated value set includes the value *above*.
- Constraints will only be **pushed** if the `hierarchicalScope` parameter enumerated value set includes the value *below*.

Note: This function will attempt to open all necessary layouts in edit mode and will build the constraint caches for all the masters where the constraint was copied to.

See also Pushing and Pulling Constraints in a Layout Hierarchy in the *Virtuoso Unified Custom Constraints User Guide*.

Arguments

u_cons	The source constraint from the lower level layout.
u_targetCache	The destination constraint cache up the design hierarchy.
?paths *l_paths*	A list of occurrence path names (the list of paths where the constraints are pulled). If not specified, the constraint will be pulled in all occurrence paths relevant by constraint members connectivity. This is an optional argument with a default of `nil`.

`?autoTag` *`g_autoTag`*	When set, if the constraint can be propagated it will be tagged with the `hierarchicalScope parameter = "above"`, and will be propagated. Otherwise, only those constraints that are already tagged will be propagated. This is an optional argument with a default of `nil`.
`?recursive` *`g_recursive`*	When set, the constraint will be pulled up as many levels as possible by constraint members connectivity in all applicable occurrence paths. This is an optional argument with a default of `nil`.
`?update` *`g_update`*	When set, occurences of the constraint in the hierarchy levels above will be updated. Otherwise, a new constraint will be created. This is an optional argument with a default of `nil`. When `nil`, and a constraint already exists but is different, that constraint will not be modified (neither updated nor deleted).

Value Returned

`t`	Constraint successfully pulled up the layout hierarchy as per settings.
`nil`	Constraint failed to be pulled.

Example

```
ciPullConstraint(con top_cache ?recursive t ?paths list("|I18/I1"))
```

This will pull all connected constraint from a lower level design (`source_cache`) to the `top_cache` along the path "`|I18/I1`". That is, it will pull the constraint from `I1` to `I18` and then recursively from `I18` to `top_cache`.

ciPullConstraints

```
ciPullConstraints(
     u_sourceCache
     u_targetCache
     [ ?paths l_paths ]
     [ ?autoTag g_autoTag ]
     [ ?recursive g_recursive ]
     [ ?update g_update ]
     )
     => constraintIdList / nil
```

Description

Copies all applicable constraints up a layout hierarchy.

This can be described as “pulling” constraints up the design hierarchy. The copied constraints’ members (nets, terminals instance terminals and so on) will also correspond to the original constraint members as they are physically connected.

- Constraints will only be **pulled** if the `hierarchicalScope` parameter enumerated value set includes the value *above*.
- Constraints will only be **pushed** if the `hierarchicalScope` parameter enumerated value set includes the value *below*.

Note: This function will attempt to open all necessary layouts in edit mode and will build the constraint caches for all the masters where the constraints are copied to.

See also Pushing and Pulling Constraints in a Layout Hierarchy in the *Virtuoso Unified Custom Constraints User Guide*.

Arguments

u_sourceCache	The source constraint cache in the lower level layout.
u_targetCache	The destination constraint cache up the design hierarchy.
?paths *l_paths*	A list of occurrence path names (the list of paths where the constraints are pulled). If not specified, the constraints are pulled in all occurrence paths relevant by constraint members connectivity. This is an optional argument with a default of `nil`.

`?autoTag` *`g_autoTag`*	When set, if the constraint can be propagated it will be tagged with the `hierarchicalScope parameter = "above"`, and will be propagated. Otherwise, only those constraints that are already tagged will be propagated. This is an optional argument with a default of `nil`.
`?recursive` *`g_recursive`*	When set, constraints will be pulled up as many levels as possible by constraint members connectivity in all applicable occurrence paths. This is an optional argument with a default of `nil`.
`?update` *`g_update`*	When set, existing constraints in the hierarchy levels above will be updated. Otherwise, only new constraints will be created. This is an optional argument with a default of `nil`.

Value Returned

`t`	Constraints successfully pulled up the layout hierarchy as per settings.
`nil`	Constraints failed to be pulled.

Example

```
ciPullConstraints(source_cache top_cache ?recursive t ?paths list("|I18/I1"))
```

This will pull all connected constraints from a lower level design (`source_cache`) to the `top_cache` along the path "`|I18/I1`". That is, it will pull constraints from `I1` to `I18` and then recursively from `I18` to `top_cache`.

ciPushConstraint

```
ciPushConstraint(
     u_constraint
     [ ?paths l_paths ]
     [ ?autoTag g_autoTag ]
     [ ?recursive g_recursive ]
     [ ?update g_update ]
     )
     => constraintIdList / nil
```

Description

Copies a single of constraint down a layout hierarchy.

This can be described as "pushing" the constraint down a design hierarchy. The copied constraint's members (nets, terminals instance terminals, and so on) will also correspond to the original constraint members as they are physically connected.

- Constraints will only be **pulled** if the `hierarchicalScope` parameter enumerated value set includes the value *above*.
- Constraints will only be **pushed** if the `hierarchicalScope` parameter enumerated value set includes the value *below*.

Note: This function will attempt to open all necessary layouts in edit mode and will build the constraint caches for all the masters where constraints are copied to.

See also Pushing and Pulling Constraints in a Layout Hierarchy in the *Virtuoso Unified Custom Constraints User Guide*.

Arguments

u_constraint	The source constraint from the top layout.
?paths *l_paths*	A list of occurrence path names (the list of paths where the constraint is to be pushed). This is an optional argument with a default of `nil`.
?autoTag *g_autoTag*	When set, if the constraint can be propagated it will be tagged with the `hierarchicalScope parameter = "below"`, and will be propagated. Otherwise, only those constraints that are already tagged will be propagated. This is an optional argument with a default of `nil`.

`?recursive` *`g_recursive`*	When set, the constraint will be pushed down as many levels as possible in all applicable occurrence paths. This is an optional argument with a default of `nil`.
`?update` *`g_update`*	When set, occurences of the constraint in the hierarchy levels below will be updated. Otherwise, only new constraints will be created. This is an optional argument with a default of `nil`.

Value Returned

`t`	Constraint successfully pushed down the layout hierarchy as per settings.
`nil`	Constraint failed to be pushed.

ciPushConstraints

```
ciPushConstraints(
     u_cache
     [ ?paths l_paths ]
     [ ?autoTag g_autoTag ]
     [ ?recursive g_recursive ]
     [ ?update g_update ]
     )
     => constraintIdList / nil
```

Description

Copies a set of constraints down a layout hierarchy.

This can be described as “pushing” constraints down a design hierarchy. The copied constraints’ members (nets, terminals instance terminals, and so on) will also correspond to the original constraint members as they are physically connected.

- Constraints will only be **pulled** if the `hierarchicalScope` parameter enumerated value set includes the value *above*.
- Constraints will only be **pushed** if the `hierarchicalScope` parameter enumerated value set includes the value *below*.

Note: This function will attempt to open all necessary layouts in edit mode and will build the constraint caches for all the masters where constraints are copied to.

See also Pushing and Pulling Constraints in a Layout Hierarchy in the *Virtuoso Unified Custom Constraints User Guide*.

Arguments

u_cache	The source cache. The constraint cache for the top level.
?paths *l_paths*	A list of occurrence path names (the list of paths where the constraints are to be pushed). This is an optional argument with a default of `nil`. This parameter enables you to push constraints to selected instances. However, the default behavior is to push constraints in all instances.

`?autoTag` *`g_autoTag`*	When set, all constraints that can be propagated will be tagged with the `hierarchicalScope parameter = "below"`, and will be propagated. Otherwise, only those constraints that are already tagged will be propagated. Default: `nil`
`?recursive` *`g_recursive`*	When set, constraints will be pushed down as many levels as possible in all applicable occurrence paths. Default: `nil`
`?update` *`g_update`*	When set, existing constraints in the hierarchy levels below will be updated. Otherwise, only new constraints will be created. Default: `nil`

Value Returned

`t`	Constraints successfully pushed down the layout hierarchy as per settings.
`nil`	Constraints failed to be pushed.

Example

- In the following example, if you specify `l_paths` argument, then constraints will be pushed to `I1` and `I2` only.

```
ciPushConstraints(cache ?paths list("I1" "I2"))
```

- In the following example, the constraints will be first pushed to `I1` from current level and then from `I1` to `I2`.

```
ciPushConstraints(cache ?paths list("I1/I2"))
```

ciRefreshCellView

```
ciRefreshCellView(
     u_cache
     )
     => u_cache / nil
```

Description

Refreshes the constraint cache in the virtual memory with the most current copy from the disk if it has changed.

Arguments

u_cache	Specifies the constraint cache.

Value Returned

u_cache	Returns a new cache ID if the constraint cache was refreshed.
nil	The constraint cache did not need a refresh.

Example

Refresh the current constraint cache if it has been updated on the disk:

```
cache = ciGetCellView()
  when( ciCacheNeedRefresh(cache)
    cache = ciRefreshCellView(cache)
  )
```

ciRegisterConstraintEditor

```
ciRegisterConstraintEditor( list(nil
     'name t_name
     'description t_description
     'constraintType t_constraintType
     'constraintParams l_constraintParams
     'editorAvailableExpression t_editorAvailableExpression
     'startEditorExpression t_startEditorExpression
     'iconName t_iconName
     'addToToolbar g_addToToolbar
     'templateTypes l_templateTypes
     'useForTemplateEdit g_useForTemplateEdit
     'useForConstraintCreation g_useForConstraintCreation
     'useForConstraintParamEdit g_useForConstraintParamEdit
     'editableConstraintParams l_editableConstraintParams
     )
     => t / nil
```

Description

Registers a constraint editor with the *Constraint Manager* assistant and, optionally, adds a button to the *Constraint Manager* toolbar. This button can be used to open the registered constraint editor. An entry is also added to the *Constraint Editor* submenu that appears on the *Constraint Manager*'s context-sensitive menu.

See also ciUnregisterConstraintEditor.

Arguments

`'name` *`t_name`*
Name of the new constraint editor.

`'description` *`t_description`*
Description of the new constraint editor.

`'constraintType` *`t_constraintType`*
Type of constraint that can be edited using this constraint editor (if you select in the *Constraint Manager* a constraint that does not match this type, the constraint editor option is grayed out).

`'constraintParams` *`l_constraintParams`*

List of constraint parameter names and values that require to be matched for the constraint editor to be able to edit a constraint in the *Constraint Manager*. If the parameter name or values do not match the the values set using this parameter, the constraint editor option is grayed out.

`'editorAvailableExpression` *`t_editorAvailableExpression`*

Expression that is evaluated to determine whether a constraint editor is available in the application currently being run. For example, this argument can be used to specify that a constraint editor is available only in the layout or schematic *Constraint Manager*. The *Constraint Comparison Report* is available only in layout.

If this expression returns `nil`, the constraint editor is not registered with the *Constraint Manager*, and the related toolbar and menu options are not displayed.

`'startEditorExpression` *`t_startEditorExpression`*

Expression that is evaluated before the constraint editor is opened. This expression can refer to a constraint variable whose value is the selected constraint when the constraint editor is run or nil if no constraint is currently selected.

`'iconName` *`t_iconName`*

Name of the `.png` file of the constraint editor icon that should be added to the *Constraint Manager* toolbar, for example `nexGenEd.png`.

`'addToToolbar` *`g_addToToolbar`*

Determines whether or not a tool button should be added to the *Constraint Manager* toolbar for the constraint editor.

`'templateTypes` *`l_templateTypes`*

List of templates that can be edited in the constraint editor.

Note: In the *Constraint Browser*, when you select a template that does not match the template types listed with this argument, the *Constraint Editor* menu in the *Constraint Manager* toolbar appears as disabled, that is, grayed out.

`'useForTemplateEdit` *`g_useForTemplateEdit`*

Determines whether the constraint editor should be launched in the following two cases:

- When a template is double-clicked in the *Constraint Browser*
- When a template parameter is edited in the *Constraint Parameter Editor* pane

Valid Values: `t` or `nil`

Note: This argument works only when the `'useCGenForEdit` argument of the ciRegisterConstraintGenerator SKILL function is not set for the constraint generator associated with the concerned template. Otherwise, double-click or parameter edit launches the associated constraint generator dialog box.

`'useForConstraintCreation` *`g_useForConstraintCreation`*

Determines whether the constraint editor should be launched whenever a constraint of the type specified by the `constraintType` argument is created using the Constraint Manager assistant.
Valid Values: `t` or `nil`

`'useForConstraintParamEdit` *`g_useForConstraintParamEdit`*

Determines whether the constraint editor should be launched when you edit a constraint parameter in the *Constraint Parameter Editor* pane.
Valid Values: `t` or `nil`

`'editableConstraintParams` *`l_editableConstraintParams`*

List of constraint parameters for which the constraint editor is displayed when one of the listed parameters is edited in the *Constraint Parameter Editor* pane.

If a constraint parameter is not specified in this list, it is edited within the *Constraint Parameter Editor* pane.

Note: The `paramName` value specified for the `'startEditorExpression` argument can be used to display a different GUI for each constraint parameter.

Value Returned

`t`	Constraint editor successfully registered.
`nil`	Constraint editor not registered.

Example

```
ciRegisterConstraintEditor(
    list(nil 'name "Module Generator"
    'description "Modgen - Layout Structure Editor"
    'constraintType "layoutStructure"
    'constraintParams list( "type=module")
    'editorAvailableExpression "isCallable('mgCreateOrEdit)"
    'startEditorExpression "mgCreateOrEdit(geGetEditCellView () constraint nil)"
    'iconName "nexGenEd"
    'addToToolbar t
    'templateTypes list("CurrentMirror" "DiffPair")
    'useForTemplateEdit t
    'useForConstraintCreation t
    'useForConstraintParamEdit t
    'editableConstraintParams list("numRows" "numCols")
    )
)
```

ciRegisterCustomDeviceFilter

```
ciRegisterCustomDeviceFilter(
     t_name
     t_func
     )
     => t / nil
```

Description

Registers a custom filter or modifies an existing filter.

Arguments

t_name	The name of the custom filter to be registered or modified.
t_func	Action to be performed on filter.

Value Returned

`t`	Custom filter successfully registered or modified.
`nil`	Command failed.

Example

```
ciRegisterCustomDeviceFilter(“myDefault” 'CstMyFilter)
```

ciRegisterNetSuperType

```
ciRegisterNetSuperType(
     t_superType
     l_subTypes
     )
     => t / nil
```

Description

This function creates a net type that automatically includes the names, regular expressions, and predicates registered in other net types. The new net type is known as a *super-type*, and the net types included in it as *sub-types*.

Note: You cannot directly register nets in a *super-type*. Register them in one of the sub-types instead.

For more information, also see the following:

- ciGetNetSubTypes
- ciGetNetSuperTypes
- ciIsNetSuperType

Arguments

t_superType	Specifies a name for the *super-type* net.
l_subTypes	Specifies a list of *sub-type* nets to include in the *super-type* net.

Value Returned

`t`	Indicates that the new *super-type* net has been created successfully.
`nil`	Indicates that the operation failed.

Example

```
;; Register "Supply" as the union of "Power" and "Ground".
ciRegisterNetSuperType("Supply" '("Power" "Ground"))
;; Register some power and ground nets.
```

```
procedure(myPowerPredicate(net netType) net->sigType == "supply")
ciRegisterNetNames("Power" '("vcc"))
ciRegisterNetRegexs("Power" '("[vV][dD][dD]"))
ciRegisterNetPredicate("Power" 'myPowerPredicate)

procedure(myGroundPredicate(net netType) net->sigType == "ground")
ciRegisterNet("Ground" '("gnd"))
ciRegisterNetRegexs("Ground" '("[vV][sS][sS]"))
ciRegisterNetPredicate("Ground" 'myGroundPredicate)

;; The "Supply" super-type contains everything in "Power" and "Ground".
ciNetNames("Supply") ; ("vcc" "gnd")
ciNetRegexs("Supply") ; ("[vV][dD][dD]" "[vV][sS][sS]")
ciNetPredicates("Supply") ; (myPowerPredicate myGroundPredicate)
```

ciRegTypeBindingParameter

```
ciRegTypeBindingParameter(
     S_typeName
     S_paramName
     )
     => t / nil
```

Description

Registers a binding parameter for a user-defined constraint type from a `config.xml` file. The parameter value must match for the constraints of that type to be considered equivalent between schematic and layout. Specifying an empty parameter name string unregisters the binding parameter.

Arguments

S_typeName	Specifies the user-defined type.
S_paramName	Specifies the parameter name.

Value Returned

`t`	The binding parameter for was successfully.
`nil`	Indicates that the operation failed.

Example

Check if the constraint type has a binding parameter:

```
ciTypeHasBindingParameter('myType)
  => nil
```

Register a binding parameter for "`myBindingParam`" for "`myType`":

```
ciRegTypeBindingParameter("myType" "myBindingParam")
```

Confirm binding parameter has been registered:

```
ciTypeHasBindingParameter('myType)
  => t
ciTypeBindingParameter('myType)
  => "myBindingParam"
```

Unregister a binding parameter by specifying an empty string:

```
ciRegTypeBindingParameter("myType" "")
ciTypeHasBindingParameter('myType)
  => nil
```

ciRemoveConstrainedPinNetsFromRails

```
ciRemoveConstrainedPinNetsFromRails(
     g_cache
     )
     => t / nil
```

Description

Deletes any constraints on pins where the net on the pin is also a member of a rail constraint because this is currently not supported by the Analog Placer.

Arguments

g_cache	The constraints cache.

Value Returned

`t`	Constraints on pins from the specified cache successfully removed.
`nil`	Constraint on pins from the specified cache not removed.

Example

```
pinCon = ciConCreate(cache 'alignment ?members '(("VDD:2" pin) ("hold:5" pin)
("inp:9" pin)))
railCon = ciConCreate(cache 'rail ?members '(("VDD" net) ("PR Boundary" boundary)
))
mapcar(lambda((mem) car(mem)) pinCon~>members)
   ("VDD:2" "hold:5" "inp:9")
mapcar(lambda((mem) car(mem)) railCon~>members)
   ("VDD" "PR Boundary")

ciRemoveConstrainedPinNetsFromRails(cache)

*WARNING* (CMGR-6140): Pin 'VDD:2' from alignment constraint 'Constr_3' is
contained within a rail constraint 'Constr_5'. This is not supported by the analog
placer and the pin member will be deleted from the alignment constraint by
precedence rules.

mapcar(lambda((mem) car(mem)) pinCon~>members)
```

```
  ("hold:5" "inp:9")
mapcar(lambda((mem) car(mem)) railCon~>members)
  ("VDD" "PR Boundary")
```

ciRemoveHierarchicalNotes

```
ciRemoveHierarchicalNotes(
     t_libName
     t_cellname
     t_viewname
     [ g_hierarchical ]
     )
     => t / nil
```

Description

Removes the existing notes from the templates and constraints for a single cellview or all along the hierarchy beginning from the given cellview.

See also ciAddHierarchicalNotes and ciUpdateHierarchicalNotes.

Arguments

t_libName	The library that contains the cellview for which notes are to be removed.
t_cellName	The cell that contains the view for which notes are to be removed.
t_viewName	The view for which notes are to be removed.
g_hierarchical	Determines whether the notes should be removed from the whole hierarchy for the specified cellview. The valid values are `t` to remove the notes from the entire hierarchy and `nil` to remove the notes only from the specified cellview.

Value Returned

`t`	Notes successfully removed for the given cellview.
`nil`	Notes could not be removed.

Example

To remove notes from the entire hierarchy starting from the specified cellview:

```
ciRemoveHierarchicalNotes("myLib" "myCellName" "myView" t)
```

ciRemoveLeadingSlash

```
ciRemoveLeadingSlash(
     t_name
     )
     => t_name
```

Description

Removes the leading forward slash from a string if it has one.

Arguments

t_name	The name from which the leading forward slash needs to be removed.

Value Returned

t_name	The name with a leading forward slash removed.

Example

```
ciRemoveLeadingSlash("/NM1")
=> "NM1"
ciRemoveLeadingSlash("I1/NM1")
=> "I1/NM1"
```

ciRemoveProcessRules

```
ciRemoveProcessRules(
     d_Id
     )
     => t / nil
```

Description

Removes all process rules associated with the given object.

Arguments

d_Id	The database identifying number for the route or net. **Note:** The `ciRemoveProcessRules` function also accepts cellviews.

Value Returned

`t`	The process rules for the given route or net have been successfully removed.
`nil`	Command failed.

Example

```
ciRemoveProcessRules(netId)
```

ciRemoveTrailingSlash

```
ciRemoveTrailingSlash(
     t_name
     )
     => t_name
```

Description

Removes the leading forward slash from a string if it has one.

Arguments

t_name	The name from which the trailing forward slash needs to be removed.

Value Returned

t_name	The name with a trailing forward slash removed.

Example

```
ciRemoveTrailingSlash("/NM1/")
=> "/NM1"
ciRemoveTrailingSlash("I1/NM1")
=> "I1/NM1"
```

ciReopenCellView

```
ciReopenCellView(
     u_cache
     t_mode
     )
     => u_cache / nil
```

Description

Reopens the constraint view in read or edit mode.

Note: This function does not apply to layout constraint caches.

Arguments

u_cache	The constraint cache.
t_mode	The mode in which the constraint view needs to be opened, that is, read (`"r"`) or edit (`"a"`).

Value Returned

u_cache	The new constraint cache ID if the mode has changed.
`nil`	The constraint cache could not be reopened in the requested mode.

Example

- Reopen current constraint cache in edit mode.

```
cache = ciGetCellView()
  when( null(ciCacheIsWritable(cache))
    cache = ciReopenCellView(cache "a")
  )
```

- Reopen current constraint cache in read mode.

```
cache = ciGetCellView()
  when( ciCacheIsWritable(cache)
    cache = ciReopenCellView(cache "r")
  )
```

ciReorderAssistants

```
ciReorderAssistants(
     l_newOrder
     )
     => t / nil
```

Description

Reorders the listing of Circuit Prospector assistants (categories). The new ordering should be a list of the existing category names in the order that you wish them to appear in the Circuit Prospector category drop-down list.

Note: A `nil` category name is interpreted as a separator in the Circuit Prospector category drop down list.

Arguments

l_newOrder	The new ordering of the Circuit Prospector assistants (categories)

Value Returned

`t`	Category rerder successful.
`nil`	Command failed.

Example

```
ciReorderAssistants(list("Nets" "Pins" "Inst Terms" nil "Devices" "Structures" nil
"Properties" nil "Electrical Constraints" "Placement Constraints" "Routing
Constraints" nil "Clusters" "Matched Parameters" "Modgens" "Orientations"
"Symmetries"))
```

This results in the *Circuit Prospector Category* drop-down list looking like:

Nets
Pins
Inst Terms

Devices
Structures

Properties

Electrical Constraints
Placement Constraints
Routing Constraints

Clusters
Matched Parameters
Modgens
Orientations
Symmetries

ciResistorArrayUpdateRowColVal

```
ciResistorArrayUpdateRowColVal(
     g_direction
     )
     => t
```

Description

Increases or decreases the row and column values of the series or parallel resistor array template that contains the currently selected modgen. If the selected topology orders the resistor devices by *Row*, the value displayed in the *Col* field is updated using this function. Conversely, if the selected topology orders the resistor devices by *Col*, the value displayed in the *Row* field is updated.

Note: This function can be helpful if bounded with a key.

Arguments

g_direction	The value of row or column is increased when set to `t` and is decreased when set to `nil`.

Value Returned

`t`	Operation successful.

Example

Set the following bindkeys:

```
hiSetBindKeys("VLS-GXL" list(list("<Btn4Down>"
"ciResistorArrayUpdateRowColVal(t)")))
```

```
hiSetBindKeys("VLS-GXL" list(list("<Btn5Down>"
"ciResistorArrayUpdateRowColVal(nil)")))
```

`Btn4Down` is scroll wheel up and `Btn5Down` is scroll wheel down. Using these bindkeys, you can increase and decrease the row or column value by scrolling mouse wheel up and down, respectively when the `figGroup` (modgen) is selected in the navigator.

ciResolveBulkNet

```
ciResolveBulkNet(
     t_device
     l_hierContext
     [ ?viewNameList g_viewNameList ]
     [ ?stopViewList g_stopViewList ]
     )
     => list / nil
```

Description

This function is used with Circuit Prospector finder expressions to retrieve the bulk net on a device and resolves the net to the highest level in the design.

If the device does not have a bulk net it will look to see whether or not the device has a property, which matches any of the bulk net property names registered with `ciMapParam("bulkParamName")`. For example, "`bulkp`" or "`bulkn`".

If a bulk property does exist, and is a net name, then the net will be resolved to the highest level in the design.

If the property is a net expression then this will be evaluated in the context of the current window's `viewNameList/stopViewList` or within the context of the passed `viewNameList/stopViewList`. Where the device symbol does not have an explicit bulk terminal, the spectre view typically has the default bulk terminal and therefore, the `viewNameList` should include spectre. For example, “schematic spectre symbol”.

Arguments

t_device	The device on which the bulk net is to be found.
l_hierContext	The current Circuit Prospector hierarchical context.
`?viewNameList` *g_viewNameList*	The `viewNameList` for evaluating net expressions. If the value is `nil`, the current window's `viewNameList` is used.
`?stopViewList` *g_stopViewList*	The `stopViewList` for evaluating net expressions. If the value is `nil`, the current window's `stopViewList` is used.

Value Returned

list	A disembodied property list (hierPath <*t_hierPath*> net <*d_netId*>) of the resolved net.
nil	A disembodied property list not found.

Example

```
ciResolveBulkNet(
     dev hierContext
     ?viewNameList "schematic spectre symbol"
     ?stopViewList "symbol"
     )
```

ciRunMatchingConstraintsGenerator

```
ciRunMatchingConstraintsGenerator(
     t_strength
     d_insts
     u_cache
     )
     => t / nil
```

Description

This SKILL function is run by the `Matching(Strength)` constraint generator expression and used to create matching template constraints.

Arguments

t_strength	Specifies level of strength (“`low`”, “`medium`”, or “`high`”). Based on this value, the `userParameter` strength of the matching template is set, for example: `` `(nil `strength "low") ``
d_insts	A list of two instances, for example: `` `(`("I1" `inst) `("I2" `inst)) ``
u_cache	The constraint cache where the matching strength template will be created.

Value Returned

`t`	Matching template strength has been set
`nil`	Command failed

Example

```
ciRunMatchingConstraintsGenerator(list(nil `strength "low") list('("MN0" inst)
'("MN1" inst)) cache)
```

where `MN0` and `MN1` are the two instances on which the matching constraint has been created.

ciSelectedConstraints

```
ciSelectedConstraints(
     g_cache
     [ ?type s_type]
     )
     => l_selectedConstraints
```

Description

Returns a list of selected constraint IDs that are contained in the passed constraints cache.

Arguments

g_cache	The constraint cache.
s_type	The constraint type. When specified, only returns the selected constraints of that type.

Value Returned

l_selectedConstraints	List of the selected constraint IDs.

Example

```
con1=ciConCreate(cache 'alignment ?members '(("NM0" inst)("NM1" inst)))
con2=ciConCreate(cache 'distance ?members '(("NM2" inst)("NM3" inst)))
hsmSelect(?type 'constraint ?name list(con1 con2))
ciSelectedConstraints(cache)~>type
   (alignment distance)
ciSelectedConstraints(cache ?type 'alignment)~>type
(alignment)
```

ciSelectedTemplates

```
ciSelectedTemplates(
     g_cache
     [ ?type s_type ]
     )
     => l_selectedTemplates
```

Description

Returns a list of selected template IDs that are contained in the passed constraints cache.

Arguments

g_cache	The constraint cache.
s_type	The constraint type. When specified, only returns the selected templates of that type.

Value Returned

l_selectedConstraints	List of the selected template IDs.

Example

```
temp1=ciTemplateCreate(cache "myTemplate1" ?members '(("NM0" inst)("NM1" inst)))
temp2=ciTemplateCreate(cache "myTemplate2" ?members '(("NM2" inst)("NM3" inst)))
hsmSelect(?type 'template ?name list(temp1 temp2))
ciSelectedTemplates(cache)~>type
   ("myTemplate1" "myTemplate2")
ciSelectedTemplates(cache ?type "myTemplate2")~>type
("myTemplate2")
```

ciSetHaloOptions

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

Description

Invokes a user interface that allows you to set the various constraint visualization options.

Arguments

None

Value Returned

`t`	Halo options successfully applied.
`nil`	Command canceled.

Example

```
ciSetHaloOptions()
```

ciSetHaloPolicy

```
ciSetHaloPolicy(
     t_haloPolicy
     )
     => t / nil
```

Description

Sets the policy for object haloing and grouping.

Arguments

t_haloPolicy

This can be either:

- `fullDisplay`

  All objects are haloed and grouped. This is the default setting.

- `noGrouping`

  Objects are only haloed. Objects are not grouped and not enclosed in a grouping box.

  Note: This mode will provide significant time savings if the number of constraint members are large.

- `groupBox`

  This option is similar to `noGrouping`, but also provides a group box to be displayed along with any un-grouped halo objects.

- `minDisplay`

  In this mode, there is a configurable limit (see ciSetMaxHaloGroupSize) to the number of database objects that get haloed and grouped together. Therefore, during constraint visualization when the number of design objects to be haloed cross a threshold, further object generation will be pruned.

Unlike `noGrouping`, where all objects are haloed but not grouped, it is possible for constraint visualization here to be incomplete and therefore inaccurate.

Value Returned

`t`	Halo policy successfully set.
`nil`	Failed to set halo policy.

Example

```
ciSetHaloPolicy("noGrouping")
```

ciSetMaxHaloGroupSize

```
ciSetMaxHaloGroupSize(
     x_maxHaloGroupSize
     )
     => t / nil
```

Description

Allows the setting of a limit on the number of database objects to be haloed and grouped together when a constraint is selected in the Constraint Manager. The setting of a group size limit is only possible when `minDisplay` has been set as the halo policy using ciSetHaloPolicy.

Arguments

x_maxHaloGroupSize	Specify the number of database objects to be haloed and grouped.

Value Returned

`t`	Maximum halo group size successfully set.
`nil`	Failed to set the maximum halo group size.

Example

```
ciSetMaxHaloGroupSize(5000) => t
```

ciSetModgenTopology

```
ciSetModgenTopology(
     g_modgen
     l_newTopology
     )
     => t / nil
```

Description

Sets the database topology on the modgen and deletes any pre-existing database topologies.

Topology objects can be created with the following functions: `dbCreateTopology`, `dbCreateTrunk`, `dbCreateTwig`, and `dbCreateStrap`. Constraints, such as `minWidth` and `validLayers`, can be applied to the topology objects using the `cstCreateConstraint` function. For further information on topology creation, refer to *Virtuoso Design Environment SKILL Reference*.

See also ciDeleteModgenTopologies.

Arguments

g_modgen	The modgen on which the topology should be set.
l_newTopology	The list of topology objects to be set on a modgen.

Value Returned

`t`	The operation was successful.
`nil`	The operation was unsuccessful.

Example

```
topology1 = dbCreateTopology(strcat("topo_" netId1->name) netId1)

trunk1 = dbCreateTrunk("trunk1" topology1 "horizontal")
dbSetTrunkAnchor(trunk1 devId1)
dbSetTrunkSide(trunk1 "top")
dbSetTrunkOrthoOffset(trunk1 0.08)

cg = cstCreateConstraintGroupOn(trunk1 "default")
```

```
cstCreateConstraint(cg "validLayers" nil list("Metal1"))
cstCreateConstraint(cg "minWidth"    list("Metal1") 0.09)

twigCnt = 0

foreach(instTerm netId1~>instTerms
    twig = dbCreateTwig(sprintf(nil "twig%d" ++twigCnt) trunk1)
    dbSetTwigObject(twig instTerm)
)

topologies = list(topology1)

ciSetModgenTopology(modgen topologies)
```

ciSetSymmetricAxes

```
ciSetSymmetricAxes(
     g_cache
     )
     => t
```

Description

Where symmetry constraints exist with multiple axes, forces a single axis of symmetry to be the most commonly used axis.

Arguments

g_cache	The constraints cache.

Value Returned

t	Symmetric axes successfully set.

Example

```
ciCacheListCon(cache)~>axis
("vertical5" "vertical1" "vertical2" "vertical3" "vertical5" "vertical5"
)
ciSetSymmetricAxes(cache)
ciCacheListCon(cache)~>axis("vertical5" "vertical5" "vertical5" "vertical5"
"vertical5" "vertical5"
)
```

ciSigTypeMatchesNetType

```
ciSigTypeMatchesNetType(
     d_net
     t_netType
     )
     => t / nil
```

Description

This function creates a net predicate that checks whether or not the `sigType` attribute of a net matches a given net type.

Note: The built-in *power* net type used by Circuit Prospector corresponds to the *supply* `sigType` used by the `database` SKILL function. Circuit Prospector uses the name *supply* as a super-type containing *power* and *ground* nets. The `ciSigTypeMatchesNetType` function takes care of this discrepancy.

For more information, see `ciRegisterNetPredicate`.

Arguments

d_net	Specifies a database net.
t_netType	Specifies a registered net type.

Value Returned

`t`	Indicates that the new net predicate has been created successfully.
`nil`	Indicates that the operation failed.

Example

```
;; Set some sigType attributes on nets
cv  = geGetEditCellView()
pwr = dbFindNetByName(cv "pwr")
gnd = dbFindNetByName(cv "gnd")
clk = dbFindNetByName(cv "clk")
pwr->sigType = "supply" ; See NOTE above.
gnd->sigType = "ground"
clk->sigType = "clock"


;; Check if sigType matches Circuit Prospector net type.
```

```
ciSigTypeMatchesNetType(pwr "power") ; t
ciSigTypeMatchesNetType(gnd "ground") ; t
ciSigTypeMatchesNetType(clk "clock") ; t
```

ciSimpleName

```
ciSimpleName(
     name_t
     type_p
     )
     => t_str
```

Description

Expands the passed name and simplifies it (removes duplicates). If the resulting name is a single string then this is returned otherwise the original name string is returned.

Arguments

name_t	The name of the string.
type_p	The type of object the name belongs to, such as `'inst`, `'net`, or `'pin`.

Value Returned

t_str	Returns a string, which is either the simplified version of the name if the name can be simplified to a single string or the original name if it cannot be simplified

Example

```
ciSimpleName("<*3>VDD" 'net)
=> "VDD"
ciSimpleName("<*2>(<*3>a,b),b,c<0:3>" 'net)
=> "<*2>(<*3>a,b),b,c<0:3>"
```

ciSortedOpenCellViews

```
ciSortedOpenCellViews(
     [ ?includeLayout g_includeLayout ]
     )
     => l_cacheIds
```

Description

Returns a list of open constraint caches from ciGetOpenCellViews sorted using ciConstraintViewLessp.

Arguments

`?includeLayout g_includeLayout`

Specifies whether the layout constraint caches are included. The default is `nil`.

Value Returned

l_cacheIds	A list of cache IDs.

Example

Sorted list of open constraint caches, including layout:

```
ciSortedOpenCellViews(?includeLayout t)
```

Report the sorted list of constraint storage library, cell, and view names:

```
mapcar('ciConstraintLCV ciSortedOpenCellViews(?includeLayout t))
    ("overview" "block1" "constraint")
    ("overview" "block1" "constrBlock")
    ("overview" "block1" "layout")
    ("overview" "block2" "constraint")
    ("overview" "middle" "constraint")
    ("overview" "penthouse" "constraint")
    ("overview" "penthouse" "layout")
    ("overview" "upper" "constraint")
```

ciTemplateAddCons

```
ciTemplateAddCons(
     u_templateId
     l_listOfConstraints
     )
     => t / nil
```

Description

Adds one or more `ci` constraints to a `ci` constraint template.

Note: This function is particularly useful during the implementation of a *Create* callback template or an *Edit* callback template. For a usage example, see `ciTemplateCreate`.

For more information, see `ciTemplateDeleteCons`.

Arguments

u_templateId	Identifies the template object to which the specified `ci` constraint needs to be added.
l_listOfConstraints	Specifies a list of constraint identifiers whose constraint should be added to the specified template.

Value Returned

`t`	Indicates that the constraints have been successfully added to the template.
`nil`	Indicates that the operation failed because either the template or one of the constraints was not valid or found.

Example

```
cache = ciGetCellView()
myTemplate = ciCreateTemplate(cache "myTemplateDefinition" "TemplateName"
list(constraint1 constraint2))
myTemplate~>constraints~>name
("constraint1" "constraint2")

extraConstraint1 = ciConCreate(cache "symmetry" list(member1 member2))
extraConstraint2 = ciConCreate(cache "shielding" list(member1 member2))
```

```
listOfExtraConstraints = list(extraConstraint1 extraConstraint2)
ciTemplateAddCons(myTemplate listOfExtraConstraints) ;; adding extraConstraint1
and extraConstraint2 to myTemplate

myTemplate~>constraints~>name ;; we should see extraConstraint1 and
extraConstraint2 names in the list below, this confirms that they have well been
added.
("constraint1" "constraint2” "extraConstraint1" "extraConstraint2")
```

ciTemplateCreate

```
ciTemplateCreate(
     u_cache
     t_templateDefinitionName
     [ ?name t_name ]
     [ ?members l_members ]
     [ ?params l_params ]
     [ ?userParams t_userParams ]
     )
     => templateId / nil
```

Description

Creates a new template.

The `ciTemplateCreate` command passes the `designObjectList` and `userParams` to the `createSymbolName` callback defined using `ciTemplateCreateDefinition`. The return value of the `checkSymbolName` call must be a list of `u_constraint`. This list of constraints is used internally to define a `ciTemplate` and manage the constraints together. Any edits of the member constraints will trigger a call to `checkSymbolName` with the list of constraints to manage the template content.

Note: See also Constraint Templates in the *Virtuoso Unified Custom Constraints User Guide*.

Arguments

u_cache	See `cache`.
t_templateDefinitionName	The template definition name. For example, "`match`" for the match template.
?name *t_name*	A name for the given template instance to be created.
?members *l_members*	Used in the same manner as constraint_member. No parameters are accepted.
?params *l_params*	List of template parameters. See Template Parameters in the *Virtuoso Unified Custom Constraint User Guide* for more details
?userParams *t_userParams*	Any SKILL object or list that the user defined callback understands.

Value Returned

templateId	The ID of the template generated.
nil	Failed to create template.

Example

```
ciTemplateCreate cache 'mypair ?members (list (list "l1" 'inst) (list "l2" 'inst))
```

```
ciTemplateCreate cache "commonCentroid" ?members '( '("A" 'inst) '("B" 'inst) '("C"
'inst) '("D" 'inst))
```

ciTemplateCreateDefinition

```
ciTemplateCreateDefinition(
     t_typeName
     s_checkSymbolName
     s_createSymbolName
     [ ?params l_params ]
     [ ?acceptsUserParams s_acceptsUserParams ]
     [ ?transferSymbolName s_transferSymbolName ]
     [ ?preDestroySymbolName s_preDestroySymbolName ]
     )
     => t / nil
```

Description

Creates a template definition.

A template is a managed collection of constraints that are created and grouped together and maintained as a collection in the Constraint infrastructure, in Virtuoso Schematic XL and Virtuoso Layout XL.

A template definition is characterized by:

- A template definition name
- Two SKILL callback (function) symbols that are called automatically:
  - The first function named the `create` callback creates the template instance
  - The second function named the `modify` callback checks the template validity while it is updated.
- Four optional arguments:
  - `?acceptsUserParams`: If set to `t`, it configures the template definition to accept user parameters. If set to `nil`, it will not allow them.
  - `?params`: Defines an optional list of template parameters. Template parameters are similar to constraint parameters and can also be accessed and manipulated by the template `create` and `modify` callback functions. For more information, refer to the *Virtuoso Unified Custom Constraints User Guide*.
  - `?transferSymbolName`: A SKILL callback symbol that is called automatically when a template is transferred to layout or to the schematic.
  - `?preDestroySymbolName`: A SKILL callback symbol that is called automatically when a template is about to be deleted.

See also, ciTemplateCreate, ciTemplateAddCons, and ciTemplateDeleteCons.

Note: See also, Constraint Templates in the *Virtuoso Unified Custom Constraints User Guide*.

Arguments

`t_typeName`	The template definition name.
`s_checkSymbolName`	The name or symbol of the SKILL callback used to check a template's instance correctness.
`s_createSymbolName`	The name or symbol of the SKILL callback used to create a list of constraints to be placed inside a new template instance.
`?params l_params`	Defines an optional list of template parameters. Template parameters are similar to constraint parameters, they allow associating parameters to a template and can also be configured and manipulated by the template `create` and `modify` callback functions.
`?acceptsUserParams s_acceptsUserParams`	Configures the template definition to accept user parameters when set to `t`. If set to `nil`, user parameters are not allowed.
`?transferSymbolName s_transferSymbolName`	Optional argument. The name or symbol of the SKILL callback that is called when a template is transferred to layout or to the schematic.
`?preDestroySymbolName s_preDestroySymbolName`	The name or symbol of the SKILL callback that is called when a template is about to be deleted.

Value Returned

`t`	The template definition successfully created.
`nil`	Failed to create template definition.

The SKILL `create` and `modify` callbacks play an important role in the template creation and update processes:

Create Callback

Purpose

Returns a list of constraints to be placed inside the new template instance.

The create callback is called when a template is instantiate from ciTemplateCreate. It is passed the list of design objects the template constraint will be created on and optionally can be passed a list of user parameters and template parameter values.

In the case of a template definition in which a list of parameter definition is specified, the create callback will typically be passed a list of template parameter values that will be fetched as default values for the relevant parameters, into the created template.

Format

```
(defun createSymbolName (g_cache l_design_objects @key (l_params nil)
(l_userParams nil))
    ;; any required user parameters will be passed through the
    ;; ciTemplateCreateDefinition into the callback
    (let (constraints)
        ;; the list of constraints to be managed by the template must be created
here
        constraints
    )
) ;; the list of constraints must be returned
```

Arguments

g_cache	The constraint cache.
l_design_objects	List containing the design objects used for the template creation. These are used to create the the template constraints on. Each design object in the list should be in the following format: `list(`*<designObjectName>* *<designObjectType>*`)` where *designObjectType* is either `'inst`, `'net`, `'pin`, or `'instTerm`.

l_params	Optional argument. Contains a list of template parameters that will pre-configure all or part of the template parameters on the created template. Each parameter in the list should be of following format: `list(`*<paramName> <paramValue>*`)` Template parameters are stored on the template and can be modified by the user after the template has been created. Template parameter modifications will trigger the template check callback and this allows the template constraints to be dynamically updated in response to the parameter changes. **Note:** This argument can only be used when at least one template parameter is defined while calling ciTemplateCreateDefinition.
l_userParams	Optional argument. Contains a list of parameters usable for any purpose. These are usually used to pass extra data to the create callback and allow more flexibility for the callback to create the template instance. Each parameter in the list should be of the following format: `list(`*<paramName> <paramValue>*`)` These parameters are not stored on the template and are used for the purpose of template creation only.

Value Returned

l_list	The list of constraints that the template will contain. If `nil` is returned, the template will not be created.

Modify Callback

Purpose

Checks the correctness of the template instance.

The `modify` callback is called whenever a template is updated by a parameter update or a constraint update, addition or deletion. It will capture the updated parameters or constraints and allow their assessment, examination, and manipulation to prevent or allow the template modification. The callback can take further actions upon the template modification on any other object in the available execution context.

Format

```
(defun checkSymbolName (u_template @key (l_oldTemplateParams nil)
(l_constraints nil) (s_modType nil) (l_userParams nil) )
    ;; any required user parameters will be passed through the
    ;; ciTemplateCreateDefinition into the callback
    (let (new_constraint_list)
    ;; the list of constraints to be managed by the template must be modified here
    new_constraint_list
    )
) ;; the list of constraints must be returned
```

Arguments

u_template	Template object which is to be updated by the modify callback.
l_oldTemplateParams	Optional argument if no template parameters are specified; mandatory otherwise. List of the modified template parameter values before the template is updated. **Note:** The list of the current (after update) template parameter values can be accessed through the template object, that is, `u_template_id->parameters`
l_constraints	Optional argument if no template parameters specified. Non-optional otherwise. List of the constraints which are either, updated, added, or deleted from the template. **Note:** The list of the current (after update) template constraints is accessible through the template object, that is, `u_template_id->constraints`

s_modType

Optional argument if no template parameters specified.

Specifies the type of update that the template is subject to.

The update type can be a parameter values update, or constraint(s) update, addition, or deletion. This symbol accepts the following value:

- `'unspecifiedChange`: In certain circumstances, the system can be unable to determinate the nature of the change when it is not one of the following.
  - `'paramUpdate`: Parameter(s) update
  - `'constraintAdd`: Constraint addition
  - `'constraintDelete`: Constraint deletion
  - `'constraintUpdate`: Constraint update

l_userParams

Optional argument. List of user parameters.

Value Returned

`t`	Allows the template update
`nil`	Disallows the template update

Transfer Callback

Purpose

Called when a template is transferred to layout or to the schematic.

Format

```
(defun transferSymbolName (u_template)
    (let ((ok t))
        printf("Template %L has been transferred (isLayout: %L)\n"
                template->name ciCacheIsLayout(template->cache))
        ok
```

```
    )
)
```

Arguments

u_template	Template object which has been transferred to the schematic or layout.

Value Returned

t	The callback ran successfully.
nil	The callback failed to run. This does not effect the transfer though.

Pre-Destroy Callback

Purpose

Called when a template is about to be deleted.

Format

```
(defun preDestroySymbolName (u_template)
    (let ((allowDestroy t))
        ;;;; Determine if the passed template is allowed to be destroyed or not
        allowDestroy
    )
)
```

Arguments

u_template	Template object which has been transferred to the schematic or layout.

Value Returned

`t`	The template was deleted successfully.
`nil`	The template deletion failed.

Example

The following two examples are covered below:

- A simple example with partial code that outlines the workflow from the creation of the template definition to the instantiation of a template.
- A complete example with complete and more complex code that fully shows how to use template callbacks and template parameters and also registering a template generator for the created template definition.

  Steps of the template creation process demonstrated:

  Step 1: Declaring the arguments passed as parameters to `ciTemplateCreateDefinion` and `ciTemplateCreate`.

  Step 2: Calling `ciTemplateCreateDefiniton` in order to create a new template definition

  Step 3: Once the new definition is available, calling `ciTemplateCreate` to instanciate a new template (from SKILL) using the definition just created.

  Step 4: (Optional) Registering a constraint generator for the template definition created, to make it usable from the constraint manager. (on top of the `ciTemplateCreate` function)

Example 1

```
;; ***********************************************************
;; Step 1: Declaring template callbacks and template parameter definitions
;; ***********************************************************

;;IMPORTANT:
```

```
;; The template definition and its callbacks and other arguments are usually defined
in .cadence/dfII/ci/generators.
;; This allows them to be loaded at the virtuoso startup

;; 1) Declaring Create Callback: 'ciMatchCreateNew
(defun ciMatchCreateNew (cache listOfDevices @key (params nil) (userParams nil))
  ;; do the necessary processing the function arguments here.
  ;; ... PROCESSING ...

 listOfconstraints ;; Then return the list of constraint to be inserted inside the
template.
)

;; 2) Declaring Modify (also called in some cases "check" ) callBack: 'ciMatchCheck
(defun ciMatchCheck (templates @key (oldTemplateParams nil) (constraints nil)
(modType nil) (userParams nil) )
 ;; Perform any necessary processing based on the value of the function arguments,
amongst which
  ;; oldTemplateParams and template~>parameters
  ;; For instance Retrieve the value of the strength parameter BEFORE update (old
value)
  paramList = exists(p oldTemplateParams car(p) == "strength")
;; …
  ;; … Do any further processing step (skipped for clarity) here
  ;; …
  ;; Retrieve  the value of the strength parameter AFTER update (current value).
  paramList = exists(p template~>parameters car(p) == "strength")

  ;; finally return nil to prevent change or t to allow them, here
 t ;; in this case return t (but it could be nil, it is enterely up to the processing
happening inside this callback)
)

;; 3) declaring a template parameter definition list
myParamsDefinitionList = list(list("strength" 'enum "low" list("low" "medium"
"high"))
  list("doubleNoDefault" 'float)
  list("doubleWDefault" 'float 3.75 2.5)
  list("doubleWDefault2" 'float 1.5 0.1 4.2)
  list("intNoDefault" 'int)
  list("intWDefault" 'int 5 1)
  list("intWDefault2" 'int 5 1 6))
```

```
;; ***********************************************************
;; Step 2: Creation of a template definition:
;; ***********************************************************
myNewTemplateDefinition = ciTemplateCreateDefinition("matchTemplateWithParameters
" ;; template name definition
  'ciMatchCreateNew        ;; modify callback function from the previous example
  'ciMatchCheck            ;; create callback function from the previous example
  ?params myParamsDefinitionList     ;; optional list of template parameter
definition
)

;; ***********************************************************
;; Step 3: Instanciation of a template that uses template parameters:
;; ***********************************************************
insts = '(("M1" inst) ("M2" inst)) ;; a simple list of instances
;; Pre-requisite: have a valid CI cache object  at this point: i.e. cache =
ciGetCellView()

;; Call to ciTemplateCreate()
myNewTemplate = ciTemplateCreate (
cache         ;; Cache storage object
"matchTemplateWithParameters"   ;; Template definition name
    ?members insts              ;; instances the template will be created on
    ?params  parameterInstances ;; list of parameter instances is passed here
    )

;; skipping step 4 for Example 1, refer to example 2
```

Example 2

```
;; ***********************************************************
;; Step 1: Declaring template callbacks and template parameter definitions
;; ***********************************************************

;;IMPORTANT:
;; The template definition and its callbacks and other
;; arguments are usually defined in .cadence/dfII/ci/generators.
;; This allows them to be loaded at the virtuoso startup

;; 1) Declaring Create Callback: 'ciMatchCreateNew
```

```
(defun ciMatchCreateNew (cache listOfDevices @key (params nil) (userParams nil))
  (let ((strength 1)
    constraints newcon
    ;; modgen configuration
    (mFactor 1) (byN 1)
    (continue t)
    strengthParam
    )

    (unless (cache && listOfDevices)
      continue = nil
      )

    strengthParam = (cadar setof(x params (car x) == "strength"))

    (cond
      ((strengthParam == "low")    strength = 1)
      ((strengthParam == "medium") strength = 2)
      ((strengthParam == "high")   strength = 3)
    )

    ;; strength must be an integer fixp checks that for us.
    (unless (fixp strength)
     (warn "Matching template example: Strenght parameter expected to be integer,
it is %L." strength)
      continue = nil
      )
    ;; matching template only works for 2 devices
    (unless ((length listOfDevices) == 2)
     (warn "Matching template example: extected a device pair (%L)" listOfDevices)
      continue = nil
     )

    (when continue
      ;; everything is set
      (warn "Matching template example: Creating Matching template with strength
%d" strength)

      (case strength
        (1
```

```
        ;; matching parameters
        newcon = (ciConCreate cache 'matchedParameters
                ?members listOfDevices
                ;;?params (list (list "matchSubset" (list "l" "w" "m")))
               ?verbose nil)
        (if newcon
          then constraints = (append1 constraints newcon)
          else (warn "Matching template example: (strength 1) failed to create
matching constraint."))
        )
       (2
        ;; matching parameters & orientation
        newcon = (ciConCreate cache 'matchedParameters
                ?members listOfDevices
                ;;?params (list (list "matchSubset" (list "l" "w" "m")))
               ?verbose nil)
        (if newcon
          then constraints = (append1 constraints newcon)
          else (warn "Matching template example: (strength 2) failed to create
matching constraint"))

        newcon = (ciConCreate cache 'relativeOrientation
                ?members listOfDevices ?verbose nil)
        (if newcon
          then constraints = (append1 constraints newcon)
          else (warn "Matching template example: (strength 2) failed to create
relative orientation constraint"))

        )


       (3
        ;; matching parameters & orientation & alignment
        newcon = (ciConCreate cache 'matchedParameters
                ?members listOfDevices
                ;;?params (list (list "matchSubset" (list "l" "w" "m")))
               ?verbose nil)
        (if newcon
          then constraints = (append1 constraints newcon)
          else (warn "Matching template example: (strength 3) failed to create
matching constraint"))

        newcon = (ciConCreate cache 'relativeOrientation
```

```
                  ?members listOfDevices ?verbose nil)
          (if newcon
            then constraints = (append1 constraints newcon)
            else (warn "Matching template example: (strength 3) failed to create
relative orientation constraint"))

          newcon = (ciConCreate cache 'alignment
                  ?members listOfDevices ?verbose nil)
          (if newcon
            then constraints = (append1 constraints newcon)
            else (warn "Matching template example: (strength 3) failed to create
alignment constraint"))

          )

         (t ;; default case
        (warn "Matching template example: Strength parameter value not recognized:
%d. No Template created"
              strength)
          )



       )
     )
     constraints ;; list of constraints to be contained in the template
     )
)

;; 2) Declaring Modify (also called in some cases "check" ) callBack: 'ciMatchCheck

(defun ciMatchCheck (templates @key (oldTemplateParams nil) (constraints nil)
(modType nil) (userParams nil) )
  (info "*********************************** \n")
  (info "Executing ciMatchCheck NEW CALLBACK \n")
  (info "*********************************** \n")
  (let (oldStrengthValue currentStrengthValue param currentConstraints members ch
template
     (continue t)
     (ret nil)
     )
    ;; We have only one template here. Therefore, let us call it a template instead.
     template = templates
```

```
ch = template~>cache
(when (ch == nil)
  (info "Error cache empty \n")
  continue = nil
  )

;; The values symbol modType can be
; 'unspecifiedChange,
; 'paramUpdate,
; 'constraintAdd,
; 'constraintDelete,
; 'constraintUpdate

;; It is important to check the change flag and take an appropriate action
;; depending on the type of change.
;; - 'constraintAdd, 'constraintDelete, 'constraintUpdate occur when one
;; or several constraints are updated or deleted or added.
;; - 'paramUpdate occurs when one or several template parameters
;; (as opposed to user parameters) are updated.
;; - 'unspecifiedChange occurs when a constraint parameters and constraints
;; have been modified at the same time. In this case, this is up to
;; this function to perform all of the required to take an appropriate action
;; if necessary.
(when continue
  (case modType
    ('paramUpdate
      (info "Info: paramUpdate case \n")

      ;; Comparing and printing the value of the parameters BEFORE and AFTER
      ;; the change that this function will (or NOT) validate.
      ;; The "oldTemplateParams" argument contains a list of the template
      ;; parameters' value BEFORE the change, that is before being edited.
      ;; This helps to 1) know which parameters are being edited
      ;; 2) compare each template parameter's old value (BEFORE changed)
      ;; to the new current one (AFTER change).
      ;; This is what we'll do here, printing the old and new current values
      ;; of these edited template parameters, so that we can see what
      ;; is the change about and then take action upon it.

      ;; Retrieve the value of the strength parameter BEFORE update (old value)
```

```
          paramList = exists(p oldTemplateParams car(p) == "strength")
          param = car(paramList)
          oldStrengthValue = caddr(param)

          ;; Retrieve  the value of the strength parameter AFTER update (current
value).
          paramList = exists(p template~>parameters car(p) == "strength")
          param = car(paramList)
          name = car(param)
          currentStrengthValue = caddr(param)

          (if (and oldStrengthValue != nil currentStrengthValue != nil)
          (info "Old value of parameter %L BEFORE update was : %L \n current value
after update is : %L \n" name
         oldStrengthValue currentStrengthValue)
          )

          ;; Retrieve and print the value BEFORE update (old value) of all the
parameters that have been changed.
        (info "oldTemplateParams length %L contains: \n" length(oldTemplateParams))
          foreach(x oldTemplateParams
            (info "The old value of param %L was %L.\n" car(x) caddr(x))
          )



         ;; Now we process the value of parameter "strength" and depending on its
value "low" "medium" or "high",
         ;; We will regenerate the constraints of the template in different ways.
          ret = t
          (when ((setof x oldTemplateParams ((car x) == "strength")) != nil)
            (info "Found that strength param has changed ! => processing its new
Value \n")
            ret = (processStrengthParameter ch template currentStrengthValue)
          )

          ret
        )

        ('constraintAdd
          ;; In the same way, we can evaluate the constraints being added to the
template
        ;; access them using the "constraints" argument, and take the appropriate
action (reject of validate the change)
```

```
          (info "Info: constraint Add case \n")
          (info "ret before add constraint = %L \n" ret)
          ret = (processAddedConstraints ch template constraints)
        )

        ('constraintDelete
         ;; In the same way, we can evaluate the constraints being deleted to the
template
         ;; access them using the "constraints" argument, and take the appropriate
action (reject of validate the change)
          (info "Info: constraint Delete case \n")
          ret = (processDeletedConstraints ch template constraints)
        )

        ('constraintUpdate
          ;; In the same way, we evaluate the constraints being updated in the
template.
        ;; IMPORTANT: In the case of an constraint update, the constraints argument
contains
          ;; a list of the old value (BEFORE change) of the constraints being
updated.
          (info "Info: constraint Update case \n")
          ret = (processUpdatedConstraints ch template constraints)
        )
        (t ;; default case
          (info "Warning: Unspecified change. \n")
          ;; TODO manual checks to determine what action is appropriate.

        )
      )
    )
  (info "Return value for ciMatchCheck = %L \n" ret)
  ret

  ) ;; let
) ;; end of the callback

;; 3) Declaring the Helpers functions used in the check callback (these allow to
simplify
   ;; the code even if these are not compulsory)
```

```
   ;; These Helpers evaluate and then take action upon updated, deleted or added
   ;; template constraints depending on the value of the strength parameter and
   ;; then allow the change or refuse it by returning t or nil.
(defun processStrengthParameter (cache template paramValue)
(let (newcon ret newcons currentConstraints)

  ;; 1) In the case of the Matching Strength Template, we need to retrieve the
  ;; current members of the constraints we are currently in the template (BEFORE
  ;; changes) and then use them as the new
  ;; members of the constraints that we will create and then add to the template.

  currentConstraints = car(template~>constraints)
  listOfDevices = currentConstraints~>members
  listOfDevices = (mapcar 'lambda( (x) list(car(x) cadr(x))) listOfDevices)
  (info "Members ok \n Members list size: %L \n" length(listOfDevices))
  foreach( mapcar x listOfDevices
    (info "member name: %L type: %L \n" car(x) cadr(x))
    (info " ")
  )

  ;; 2) We delete all of the current constraints from the template (before
  ;; creating and adding some new ones)
   currentConstraints = template~>constraints

   ;; Allows deleting a constraint from a template.
   ;; In this case we remove from template "template" the constraints contained
   ;;in list "currentConstraints"
   ;; also the t flag is optional but specifies that the constraint is not only
   ;; deleted from the template but will also
   ;; be removed from the cellview. If the flag is nil then the constraint is only
   ;; deleted from the template and remains separately in the cellview.
  (ciTemplateDeleteCons template currentConstraints t)

   (case paramValue
    ("low"
    (info "case strength = low \n")
     ;; 3) creating + adding a new constraint to the template
     newcon = (ciConCreate cache 'matchedParameters
              ?members listOfDevices
             ?verbose nil
     )
     newcons = list(newcon)
    )
```

```
    ("medium"
      (info "case strength = medium \n")
       ;; 3) creating + attaching new constraint to be added
       newcon = (ciConCreate cache 'matchedParameters
                ?members listOfDevices
                ?verbose nil
                )

       (if newcon
         then newcons = (append1 newcons newcon)
        else (warn "Matching Strength template: (strength medium) failed to create
matching constraint"))

       newcon = (ciConCreate cache 'relativeOrientation
                ?members listOfDevices ?verbose nil)
       (if newcon
         then newcons = (append1 newcons newcon)
        else (warn "Matching Strength template: (strength medium) failed to create
relative"))

      ret = t
    )

    ("high"
      (info "case strength = high \n")
       ;; 3) creating + attaching new constraint to be added
       newcon = (ciConCreate cache 'matchedParameters
                ?members listOfDevices
                ?verbose nil
                )
      (if newcon
         then newcons = (append1 newcons newcon)
         else (warn "Matching Strength template: (strength high) failed to create
matching constraint"))

      newcon = (ciConCreate cache 'relativeOrientation
                ?members listOfDevices ?verbose nil)
      (if newcon
         then newcons = (append1 newcons newcon)
         else (warn "Matching Strength template: (strength high) failed to create
relative"))
```

```
       newcon = (ciConCreate cache 'alignment
             ?members listOfDevices ?verbose nil)
      (if newcon
       then newcons = (append1 newcons newcon)
       else (warn "Matching Strength template: (strength high) failed to create
alignment constraint"))

      ret = t
      )
      (t
        (warn "Strength parameter value not recognized: %d. \n"
             value)
        ret = nil ;; will prevent the current change to happen.
      )
    ) ; case

    (if (length newcons) > 0

      ;; Allows adding a new constraints to a template.
      (ciTemplateAddCons template newcons)
    )
    (info "successfully added new con \n")
    ret
  );let
)

(defun processAddedConstraints (cache template constraints)
  (let ((re t))
    (info "This is the list of constraints to be ADDED: \n")
    dumpConstraints(constraints)

    ;; TODO here : check the value of the constraints to be added
    ;; i.e return nil to prevent the change to happen otherwise return t.

    ;; example: preventing constraints of type alignment to be added.
    (foreach con constraints
      (if (eq con~>type 'alignment) then
        re = nil ;; this is going to prevent the current change (add constraints
to the template to happen)
      )
```

```
    )

    ;;return value
    (info "Return value for adding constraints = %L \n" re)
    re
  )
)

(defun processDeletedConstraints (cache template constraints)
  (info "This is the list of constraints to be DELETED from the template: \n")
  dumpConstraints(constraints)
 ;; TODO likewise as per processAddedConstraints() function, take action using the
"constraints" parameter list
  t
)

(defun processUpdatedConstraints (cache template constraints)
  (info "This is the list of constraints to be UPDATED from the template: \n")
    dumpConstraints(constraints)
 ;; TODO likewise as per processAddedConstraints() function, take action using the
"constraints" parameter list
  t
)

;; Dump function to make obvious what constraints are updated, delete or added by
the modify callback
;; Note: in a real world scenario this  kind of function is not compulsory but can
;; help debugging and understanding the callback actions.
;; It is advisable to be generous on debug statements and dumping functionalities
in order
;; to ease the adoption of new virtuoso functions or
;; features like the template parameters within the context of template callbacks.
(defun dumpConstraints (constraints)
  (foreach mapcar con constraints
    (info "******************************************************\n")
    (info "Constraint Name: %L type: %L \n" con~>name con~>type)
    listOfDevices = (mapcar 'lambda( (x) list(car(x) cadr(x))) con~>members)
    (info "%L Members found: \n" length(con~>members))
    (foreach mapcar x listOfDevices
      (info " - name: %L type: %L \n" car(x) cadr(x))
    )
    (info "******************************************************\n")
```

```
  )
)

;; 4) Declaring the parameter definitions collection:

;;strength: enumeration parameter can take values "low" "medium" "high" and its default is "low"
myParamDefinitionList =  list(list("strength" 'enum "low" list("low" "medium" "high"))
  list("doubleNoDefault" 'float)                ;; Float parameter with no default or range
  list("doubleWDefault" 'float 3.75 2.5)     ;; Float parameter with default value 3.75 and minimum value range of 2.5
  ;; Float parameter with default value 3.75 and minimum value range of 0.1 and maximum 4.2
  list("doubleWDefault2" 'float 1.5 0.1 4.2)
  list("intNoDefault" 'int)           ;; Float parameter with no default or range
  list("intWDefault" 'int 5 1)         ;; Float parameter with default value 5 and minimum value range of 1
  ;; Float parameter with default value 5 and minimum value range of 1 and maximum 6
  list("intWDefault2" 'int 5 1 6))

;; Note: For more details on template parameters see the constraint customisation guide.

;; *********************************************************
;; Step 2: Creation of a template definition:
;; *********************************************************
myNewDefinition = ciTemplateCreateDefinition("matchNewCreateCB" ;; template name definition
        'ciMatchCheck                  ;; modify callback
        'ciMatchCreateNew               ;; create callback
      ?params myParamDefinitionList        ;; optional list of template parameter definitions
       ?acceptsUserParams t)            ;; optional flag that specifies if user parameter are allowed

;; Note: even though the names are close template parameters and user parameters are different.
;; Template parameter like constraint parameter are displayed and can be modified from the constraint manager
;; User parameters are only used to extend the definition of a parameter for specific purpose independently from the ;; Constraint Manager.

;; *********************************************************
```

```
;; Step 3: Instanciation of a template that uses template parameters:
;; ***********************************************************

;; The arguments for ciTemplateCreate are:
;; The list of paremeter does not include all of the parameters in the
;; paremeter definition used in ciTemplateCreateDefinition() because this SKILL
;; function overrides the default values defined by parameter definitions for each
;; parameter of the template. Therefore, the list does not necessarily have to
;; provide a parameter value for each parameter definition. It is a simple list of
;; parameter values:

params = list(
      list("strength" "medium")
      list("doubleNoDefault" 5.6)
      list("doubleWDefault" 2.4)
      list("intWDefault" 2)
      list("intWDefault2" 3))
    )
insts = '(("M1" inst) ("M2" inst)) ;; a simple list of instances

;; Pre-requisite: have a valid CI cache object  at this point,
;; that is, cache = ciGetCellView()
myNewTemplate = (ciTemplateCreate cache "matchNewCreateCB" ?name
"myNewTemplateName"
            ?members insts
            ?params  params ;; params
          )

;; ***********************************************************
;; Step 4: (optional) Registering a template generator for the template definition
;; just created.
;; ***********************************************************

;; add the constraint generator in the browser make sure you have an
;; icon that exists; this example uses the "add" icon; put your icon
;; in .cadence/dfII/icons
ciRegisterConstraintGenerator(
list(nil
  'name "My Demo generator"
  'description "Generate Various Levels of Constraints for a pair"
  'expression "(demoNewPair args insts cache)"
```

```
  'addToToolbar t
  'iconName "add"
  'args list(
    list( "strength" `enum "low" "medium" "high")
    )
  )
)

ciRegisterConstraintGenerator(
     list(nil
    'name           "newMatchTplCreateCB"
    'description    "Creates a Match Strength Template with template parameters"
    'expression     "ciRunMatchingConstraintsGeneratorNewCreateCB(args insts 
cache)"
            'addToolbar   t
            'iconName       "templateMatched"
            'args           list(
                                   list( "strength" `enum "low" "medium" "high")
                                )
     )
  )

;; Function that gets call by the constraint generator when the UI button gets 
pressed to generate a template
 (defun ciRunMatchingConstraintsGeneratorNewCreateCB (args insts cache)
  let( (ret strength)

    if( length(insts) != 2 then
    ;; The number of instances has to be 2 in the case of this template.
      (info "Select 2 instances to create a template")
    else

  ;; putting together a strength parameter value depending on the value of the 
strength argument
  ;; of the constraint generator.
  ;; This is the typical way a constraint generator argument often interacts with
  ;; template parameters.
  (case args->strength
    ("low"
    params = (list (list "strength" "low"))
     )
    ("medium"
```

```
     params = (list (list "strength" "medium"))
     )
    ("high"
     params = (list (list "strength" "high"))
     )
    (t
    params = (list (list "strength" "low"))
     )
    )

  ret = (ciTemplateCreate cache ;; the cache storage i.e can retrieve for the 
current cellview using ciGetCellView()
     "matchNewCreateCB" ;; template definition name (the one created in this 
example)
     ?name "myMatchTemplateWithTemplateParameters" ;; optional: name of the 
template, automatically generated
     otherwise
     ?members insts ;; instances on which the template is created
     ?params  params) ;; template parameter values passed, in this case only 
strength
    )
    ret
  )
)
```

ciTemplateCreateExpanded

```
ciTemplateCreateExpanded(
     u_cache
     t_templateDefinitionName
     [ ?name t_name ]
     [ ?members l_members ]
     [ ?params g_params ]
     [ ?userParams g_userParams ]
     [ ?compress g_compress ]
     )
     => templateId / nil
```

Description

The same functionality as ciTemplateCreate, but first expands any iterated or repeated members. For example, `MN1<0:2>` gets expanded to `MN1<0>`, `MN1<1>`, and `MN1<2>`.

In the case of net repetitions, such as `<*3>inp`, by default these get expanded `N` times as `inp`, `inp`, `inp`. However, if the optional compress argument is set to `t`, then `<*3>inp` will be expanded to just `inp`. The same applies to net bundles that contain repetitions. For example `<*3>inp,a,b,c,inp,inp` will by default be expanded to `inp,inp,inp,a,b,c,inp,inp` but with compress set to `t` will be expanded to `inp,a,b,c`.

Arguments

u_cache	The constraint cache. See `cache`.
t_templateDefinitionName	The template definition name.
?name *t_name*	A name for the given template instance to be created.
?members *l_members*	Specify the iterated or repeated member.
?params *l_params*	List of template parameters. See Template Parameters in the *Virtuoso Unified Custom Constraint User Guide* for more details
?userParams *g_userParams*	Any SKILL object or list that the user defined callback understands.
?compress *g_compress*	When set to `t`, any repetitions will only be expanded to a single bit rather than `N` bits.

Value Returned

templateId	The template definition successfully created.
nil	Failed to create template definition.

Example

```
instsNetsPins = list( list("MN1<4:0>" 'inst) list("MN0" 'inst) list("<*3>inp" 'net)
)
temp1 = ciTemplateCreateExpanded(cache "MyTemplate" ?name "temp1" ?members
instsNetsPins ?compress t)


mapcar( lambda( (mem) car(mem) ) temp1~>members)
("MN1<4>" "MN1<3>" "MN1<2>" "MN1<1>" "MN1<0>" "MN0" "inp")
```

ciTemplateDefinitionExists

```
ciTemplateDefinitionExists(
     t_templateDefinitionName
     )
     => templateId / nil
```

Description

Verifies whether a template definition with a given name exists.

Arguments

t_templateDefinitionName	The template definition name.

Value Returned

templateId	The template definition.
nil	No template definition exists.

Example

```
cache = ciCacheGet("amsPLL" "vco" "schematic")cache->templates

(ci:0x181c81f8 ci:0x1842a2a8)

cache->templates~>name

("MatchTemplate0" "MatchTemplate1")

ciTemplateFind(cache "MatchTemplate0")

ci:0x181c81f8

cache->templates~>??

(((cache ci:0x18144680)
          type("CurrentMirror")
          (name "MatchTemplate0")
          (constraints
              (ci:0x17597978 ci:0x1776dee8)
          )
     )
     ((cache ci:0x18144680)
          type("match")
          (name "MatchTemplate1")
```

```
          (constraints
              (ci:0x15a17070)
          )

    )

)

ciTemplateDefinitionExists("match")

t
```

ciTemplateDelete

```
ciTemplateDelete(
     d_templateId
     )
     => t / nil
```

Description

Deletes given template.

Arguments

d_templateId	The name of the template to be deleted.

Value Returned

t	Template successfully deleted.
nil	Template not deleted.

ciTemplateDeleteCons

```
ciTemplateDeleteCons(
     u_templateId
     l_listOfConstraints
     [ s_doDelete t | nil ]
     )
     => t / nil
```

Description

This function deletes or removes one or more CI constraints from a CI constraint template.

By default, the `s_doDelete` optional argument is set to `t`. Setting `s_doDelete` to `t` detaches the constraint specified by `l_listOfConstraints` from the template object specified by `u_templateId`, and then deletes it.

On the contrary, if `s_doDelete` is set to `nil`, the specified constraint will only be detached from the template without being deleted from the cellview it belongs to. It will no longer be a part of the template, but will continue to be visible in the cellview.

Note: This function is particularly useful during the implementation of a *Create* callback template or an *Edit* callback template. For a usage example, see `ciTemplateCreate`.

For more information, see `ciTemplateAddCons`.

Arguments

u_templateId	Specifies the template object from which one or more CI constraints need to be deleted.
l_listOfConstraints	Specifies a list of constraints that need to be detached from the template and/or deleted from the current design if the `s_doDelete` flag is set to `t`.
`s_doDelete t	nil`	This is a Boolean flag which if set to `t` (default value) will not only cause the function to detach the specified constraints from the specified template but will also delete the constraints from the design it belongs to. If set to `nil`, the constraints will only be detached from the template and will remain as separate constraints in the design they belong to.

Value Returned

`t`	Indicates that the constraints have been successfully deleted from the specified template.
`nil`	Indicates that the operation failed because either the template or one of the constraints was not valid or found.

Example

The following example illustrates a situation when `s_doDelete` is set to `t`:

```
cache = ciGetCellView() ;; get the current constraint cellview, which contains all
the template and constraints for the current design.
myTemplate = ciCreateTemplate(cache "myTemplateDefinition"
"TemplateName"list(constraint1 constraint2 constraint3))
myTemplate~>constraints~>name
("constraint1" "constraint2")


listOfConstraintsToDelete = list(constraint1 constraint2)
ciTemplateDeleteCons(myTemplate listOfConstraintsToDelete) ;; doDelete defaulted
to t, this will cause the constraint to be deleted from its design.


myTemplate~>constraints~>name
("constraint3")
```

The following example illustrates a situation when `s_doDelete` is set to `nil`:

```
cache~>constraints~>name
("constraint1" "constraint2" "constraint3" "constraint4")


myTemplate = ciCreateTemplate(cache "myTemplateDefinition"
"TemplateName"list(constraint1 constraint2 constraint3 constraint4))
myTemplate~>constraints~>name
("constraint1" "constraint2" "constraint3" "constraint4")
listOfConstraintsToDelete = list(constraint1 constraint2)
ciTemplateDeleteCons(myTemplate listOfConstraintsToDelete) ;; doDelete defaulted
to t will cause the constraint deletion from cache (not only detach it from
myTemplate)


myTemplate~>constraints~>name
("constraint3" "constraint4")
cache~>constraints~>name
"constraint3" "constraint4")
;; Now, set doDelete to nil so that it only detaches the constraints without
deleting them
listOfConstraintsToDetach = list(constraint3)
ciTemplateDeleteCons(myTemplate listOfConstraintsToDetach ?doDelete nil) ;;
```

```
doDelete set to nil will only detach the selected constraints from the template
only (does not delete the constraint from cache)
myTemplate~>constraints~>name ;; we can see that only constraint4 remains in the
template("constraint4")


cache~>constraints~>name
"constraint3" "constraint4") ;; and also constraint3 has not been deleted from the
design, only detached from its template.
```

ciTemplateFind

```
ciTemplateFind(
     u_cache
     t_templateName
     )
     => t / nil
```

Description

Finds template in cache.

Arguments

u_cache	The constraint cache. See cache
t_templateName	The name of the template to be found.

Value Returned

t	Template found successfully.
nil	Template not found.

Example

```
cache = ciCacheGet("amsPLL" "vco" "schematic")
cache->templates
(ci:0x181c81f8 ci:0x1842a2a8)
cache->templates~>name
("MatchTemplate0" "MatchTemplate1")
ciTemplateFind(cache "MatchTemplate0")
ci:0x181c81f8
```

In this example, the corresponding template ID is found based on the specified template name.

ciTemplateGetCache

```
ciTemplateGetCache(
     u_templateId
     )
     => cache_ID
```

Description

Returns the constraint cache that contains the template.

Arguments

u_templateId	Specifies the ID of the template whose constraint cache you want to return.

Value Returned

cache_ID	The ID of the constraint cache.

Example

```
ciTemplateGetCache(templateId) => cacheId
```

ciTemplateGetComment

```
ciTemplateGetComment(
     u_templateId
     )
     => t_templateCommentStatus / nil
```

Description

Returns the comment that shows the template status.

Arguments

u_templateId	Specifies the ID of the template for which you want to return the status comment.

Value Returned

t_templateCommentStatus	The template status comment.
nil	Invalid template ID; therefore, template status comment could not be returned.

Example

```
cache = ciGetCellView()
tpl = car(cache~>templates)
comment = ciTemplateGetComment(tpl)
print("%s" comment)
```

Prints the template status comment for the given template ID.

ciTemplateGetCreatedTime

```
TemplateGetCreatedTime(
     u_templateId
     )
     => x_template_created_time / nil
```

Description

This function returns the creation time of a template, which is the time and date close to a second when the template has been created.

Note: The *u_template_id* must be a legal reference to a template. Using a *u_template_id* of a template, which has been deleted can result in a fatal error.

Arguments

u_templateId	Specifies the ID of the template whose creation time you want to return.

Value Returned

x_template_created_time	Returned if the specified template ID is valid. It is the time when the constraint was created, of type integer.
nil	Returned if the template is not found.

Example

Pre-requisite: Create a template (see documentation for ciTemplateCreate).

```
cache = ciGetCellView()
cache~>templates~>name
("myTemplate")


tpl = car(cache~>templates)
tpl~>name
"myTemplate"


ciTemplateGetCreatedTime(tpl)
1344335532
```

ciTemplateGetDefName

```
ciTemplateGetDefName(
     d_templateId
     )
     => templateDefinitionName / nil
```

Description

Gets the template definition name.

Arguments

d_templateId	The ID of the template whose definition name is to be got.

Value Returned

templateDefinitionName	The template definition name.
nil	Template definition name not found.

ciTemplateGetName

```
ciTemplateGetName(
     d_templateId
     )
     => templateName / nil
```

Description

Gets the template name.

Arguments

d_templateId	The ID of the template whose name is to be got.

Value Returned

templateName	The template name.
nil	Template name not found.

ciTemplateGetNote

```
ciTemplateGetNote(
     u_template_id
     )
     => t_note / nil
```

Description

Returns the note parameter of a template.

Note: The *u_template_id* must be a legal reference to a template. Using a *u_template_id* of a template which has been deleted can result in a fatal error.

For more information, see ciTemplateSetNote.

Arguments

u_template_id	Specifies the ID of the template object.

Value Returned

t_note	Displays the details of the note if it exists for the given template.
nil	Returned if template note is not found.

Example

Pre-requisite: Create a template (see documentation for ciTemplateCreate).

```
cache = ciGetCellView()
cache~>templates~>name
("myTemplate")


tpl = car(cache~>templates)
ciTemplateGetNote(tpl)
""
ciTemplateSetNote(tpl "testNote") ;; the note is currently empty "". So, assign a
new value for the template note


ciTemplateGetNote(tpl)
"testNote"
```

ciTemplateGetStatus

```
ciTemplateGetStatus(
     u_templateId
     )
     => t_templateStatus / nil
```

Description

Returns the template status.

Arguments

u_templateId	Specifies the ID of the template for which you want to return the status.

Value Returned

t_templateStatus	The template status.
nil	Invalid template ID; therefore, template status could not be returned.

Example

```
cache = ciGetCellView()
tpl = car(cache~>templates)
ciTemplateGetStatus(tpl)
```

Prints the template status for the given template ID.

ciTemplateGetType

```
ciTemplateGetType(
     u_template_id
     )
     => t_template_type_name / nil
```

Description

Returns the symbolic name of the given template type, such as `commonCentroid`.

Note: The *`u_template_id`* must be a legal reference to a template. Using a *`u_template_id`* of a template, which has been deleted can result in a fatal error.

For more information, see `ciTemplateSetNote`.

Arguments

`u_template_id`	Specifies the ID of the template whose symbolic name you want to return.

Value Returned

`t_template_type_name`	Displays the symbolic name (textual value) of the template type.
`nil`	Returned if no symbolic name is found for the given template type.

Example

Pre-requisite: Create a template (see documentation for `ciTemplateCreate`).

```
cache = ciGetCellView()
cache~>templates~>name
("myTemplate")

tpl = car(cache~>templates)
ciGetTemplateType(tpl) ;; get the name of the symbol defining the template type
"commonCentroid"

tpl~>type              ;; you can equally get the type by using the ~> operator
with the type property

"commonCentroid"
```

ciTemplateListCon

```
ciTemplateListCon(
     d_templateId
     )
     => constraintIdList / nil
```

Description

Gets the template constraint list.

Arguments

d_templateId	The id of the template whose constraint list is to be got.

Value Returned

constraintIdList	The template constraint list.
nil	Template constraint list not found.

ciTemplateListParamNames

```
ciTemplateListParamNames(
     u_template_id
     )
     => l_paramNames_list / nil
```

Description

Returns a list of template parameter names if the template has a list of parameters in its definition.

Note: The *u_template_id* must be a legal reference to a template. Using a *u_template_id* of a template, which has been deleted can result in a fatal error.

For more information, see `ciTemplateListParams`.

Arguments

u_template_id	Specifies the ID of the template whose parameter names you want to get.

Value Returned

l_paramNames_list	Displays a list of parameter names from the definition of the given template.
`nil`	Returned if no parameters are found in the definition of the given template.

Example

Pre-requisite: Create a template (see documentation for `ciTemplateCreate`).

```
cache = ciGetCellView()
cache~>templates~>name
("myCurrentMirror")

tpl = car(cache~>templates)
tpl~>type
"CurrentMirror"

ciTemplateListParamNames(tpl)
    ("Style" "Add Dummies" "Abut All" "Add GuardRing" "Type"
    "Shape" "Net" "Spacing" "Left Spacing" "Right Spacing"
    "Top Spacing" "Bottom Spacing" "MPP" "Use Fluid" "Device"
```

```
    "Width" "Use Min DRC for Spacing"
)
```

ciTemplateListParams

```
ciTemplateListParams(
     u_template_id
     [ s_includeDefaults_bool ]
     )
     => l_paramNames_list / nil
```

Description

Returns a list of template parameter name, type, and value, provided the template has a list of parameters in its definition.

Arguments

u_template_id	Specifies the template object ID for which we want to retrieve the parameter names and values.
s_includeDefaults_bool	Specifies a Boolean value (`t` if unspecified) that includes or excludes parameters with default values from the returned list of parameters. When set to `t` includes the default parameters, when set to `nil` excludes them.

Value Returned

l_paramNames_list	Returns a list of template parameter name, type, and value of the following form: `( (name type value) (name type value) ... (name type value))`
`nil`	Returns nil if the chosen template does not have parameters in its definition.

Example

```
/*Pre-requisite for this function is to create a template (see documentation for ciTemplateCreate())*/
cache = ciGetCellView()

cache~>templates~>name
("myCurrentMirror")
```

```
tpl = car(cache~>templates)
tpl~>type
"CurrentMirror"

ciTemplateListParams(tpl)
(("Style" enum "SingleRow")
    ("Add Dummies" boolean t)
    ("Abut All" boolean t)
    ("Add GuardRing" boolean nil)
    ("Type" enum "ring")
    ("Shape" enum "rectangular")
    ("Net" enum "inp")
    ("Spacing" float 0.0)
    ("Left Spacing" float 0.0)
    ("Right Spacing" float 0.0)
    ("Top Spacing" float 0.0)
    ("Bottom Spacing" float 0.0)
    ("MPP" enum "N-Tap")
    ("Use Fluid" boolean nil)
    ("Device" enum "NM1")
    ("Width" float 0.0)
    ("Use Min DRC for Spacing" boolean nil)
)

ciTemplateListParams(tpl nil) ;; set the includeDefaults flag to nil to exclude all
parameters with default values.
(("Style" enum "SingleRow")
    ("Add Dummies" boolean t)
)
```

Reference

ciTemplateListParamNames

ciTemplatep

```
ciTemplatep(
     g_value
     )
     => t / nil
```

Description

Checks if an object is a valid constraint template ID.

Arguments

g_value	Specifies the object to be checked.

Values Returned

`t`	Returns `t` if the specified value is a valid constraint ID.
`nil`	Returns `nil` if the specified value if not a valid constraint ID.

Example

```
when( ciTemplatep(obj) info("Object is a valid constraint template %L\n" obj) )
```

ciTemplateResetAllParams

```
ciTemplateResetAllParams(
     u_template_id
     )
     => t / nil
```

Description

Resets all template parameters to their default values.

Arguments

u_template_id	Specifies the object ID of the template whose parameters will be reset to their default value.

Values Returned

`t`	All parameters are successfully reset to their default value.
`nil`	An error occurred during execution.

Example

```
Pre-requisite is to create a template, see documentation for ciTemplateCreate()
cache = ciGetCellView()

cache~>templates~>name
("myCurrentMirror")

tpl = car(cache~>templates)
tpl~>type
"CurrentMirror"

ciTemplateListParams(tpl) ;; list all parameters including default parameters
(("Style" enum "SingleRow")
    ("Add Dummies" boolean t)
    ("Abut All" boolean t)
    ("Add GuardRing" boolean nil)
    ("Type" enum "ring")
    ("Shape" enum "rectangular")
```

```
    ("Net" enum "inp")
    ("Spacing" float 0.0)
    ("Left Spacing" float 0.0)
    ("Right Spacing" float 0.0)
    ("Top Spacing" float 0.0)
    ("Bottom Spacing" float 0.0)
    ("MPP" enum "N-Tap")
    ("Use Fluid" boolean nil)
    ("Device" enum "NM1")
    ("Width" float 0.0)
    ("Use Min DRC for Spacing" boolean nil)
)

;; set the includeDefaults flag to nil to exclude all parameters with default
values.
;; This show us the only parameters for which the value is not the default value.
ciTemplateListParams(tpl nil)
(("Style" enum "SingleRow") ;; default value is "Auto"
    ("Add Dummies" boolean t) ;; default value is nil
)

ciTemplateResetAllParams(tpl) ;; resets all of the parameters to their default
value, in this case will reset "Style" and "Add Dummies" parameters.
t

ciTemplateListParams(tpl nil) ;; Now try listing non default value parameters ONLY.
nil                           ;; The result is nil since all parameters are now
reset to their default value

ciTemplateListParams(tpl t) ;; list all parameters including default parameters
(("Style" enum "Auto")         ;; We can now see "Auto" and "Add Dummies" back to
their default value
    ("Add Dummies" boolean nil)
    ("Abut All" boolean t)
    ("Add GuardRing" boolean nil)
    ("Type" enum "ring")
    ("Shape" enum "rectangular")
    ("Net" enum "inp")
    ("Spacing" float 0.0)
    ("Left Spacing" float 0.0)
    ("Right Spacing" float 0.0)
    ("Top Spacing" float 0.0)
```

```
    ("Bottom Spacing" float 0.0)
    ("MPP" enum "N-Tap")
    ("Use Fluid" boolean nil)
    ("Device" enum "NM1")
    ("Width" float 0.0)
    ("Use Min DRC for Spacing" boolean nil)
)
```

Reference

- ciTemplateListParams
- ciTemplateListParamNames

ciTemplateResetParams

```
ciTemplateResetParams(
     u_template_id
     l_listOfParamNames
     )
     => t / nil
```

Description

Resets one or more template parameters to their default value.

Arguments

u_template_id	Specifies the ID for the template whose parameters need to be reset to their default value.
l_listOfParamNames	Specifies a list (string) containing the parameter names of the parameter to reset.

Values Returned

t	All parameters are successfully reset to their default values.
nil	An error occurs during execution.

Example

```
;;Pre-requisite to create a template, see documentation for ciTemplateCreate()
cache = ciGetCellView()

cache~>templates~>name
("myCurrentMirror")

tpl = car(cache~>templates)
tpl~>type
"CurrentMirror"

ciTemplateListParams(tpl t) ;;listing all parameters including default values
parameters
(("Style" enum "SingleRow")
    ("Add Dummies" boolean t)
```

```
    ("Abut All" boolean t)
    ("Add GuardRing" boolean nil)
    ("Type" enum "ring")
    ("Shape" enum "rectangular")
    ("Net" enum "3")
    ("Spacing" float 0.0)
    ("Left Spacing" float 0.0)
    ("Right Spacing" float 0.0)
    ("Top Spacing" float 0.0)
    ("Bottom Spacing" float 0.0)
    ("MPP" enum "N-Tap")
    ("Use Fluid" boolean nil)
    ("Device" enum "C0")
    ("Width" float 0.0)
    ("Use Min DRC for Spacing" boolean nil)
)

ciTemplateListParams(tpl t) ;; listing only non default valued parameters.
(("Style" enum "Auto")
    ("Add Dummies" boolean t)
)

ciTemplateResetParams(tpl list("Style")) ;; reseting the "Style" parameter
t

ciTemplateListParams(tpl t) ;; checking that the "Style" parameter has been reset
and that only the "Add Dummies" parameter remains.
(
  ("Add Dummies" boolean t)
)
```

Reference

ciTemplateResetAllParams

ciTemplateSetNote

```
ciTemplateSetNote(
     u_template_id
     t_note
     )
     => t / nil
```

Description

Sets the note parameter of a template.

Arguments

u_template_id	Specifies the ID of the template for which you want to setup a new note value.
t_note	Specifies the note value that will be affected to the selected template.

Values Returned

t	The note was successfully set.
nil	No template note was found or the function encountered an error during execution.

Example

```
;;Pre-requisite is to create a template, see documentation for ciTemplateCreate()
cache = ciGetCellView()
cache~>templates~>name
("myTemplate")
tpl = car(cache~>templates)
ciTemplateGetNote(tpl)
""
ciTemplateSetNote(tpl "testNote") ;; the note is currently empty "" so we assign a new value for the template note
ciTemplateGetNote(tpl)
"testNote"
```

Reference

ciTemplateGetNote

ciTemplateSetStatus

```
ciTemplateSetStatus(
     u_templateId
     s_statusSymbol
     t_statusComment
     )
     => t / nil
```

Description

Sets the template status.

Arguments

u_templateId	Specifies the ID of the template whose status you want to set.
s_statusSymbol	Specifies the symbol of the template status.
t_statusComment	Specifies the template status comment.

Value Returned

t	The template status was set successfully.
nil	The template status was not set.

Example

```
cache = ciGetCellView()

tpl = car(cache~>templates)

ciTemplateSetStatus(tpl 'enforced "Enforced")
```

Sets the template status and comment for the given template ID.

ciTemplateSortParamDefs

```
ciTemplateSortParamDefs(
     u_template_id
     )
     => l_sortedParamDef / nil
```

Description

Sorts parameter definitions of a template type, by name and returns a list of parameter definitions sorted in alphabetical order.

Arguments

u_template_id	Specifies the template ID for which we want to sort the parameter definitions.

Values Returned

l_sortedParamDef Returns a list of parameter definitions sorted by name in a ascending order.

The list is a list of parameter definition. Each definition is a list of the following form: `(definitionName dataType defaultValue)`

Parameter definition can also feature an expression, which is a SKILL expression whose evaluation result defines the parameter definition default value, enum choices or value range, where applicable.

Examples of parameter definition without expression:

- `("myIntegerParamDef" int 5)`

  In this example, "`myIntegerParamDef`" is the parameter name, `int` is the data type and `5` is the default value.

- `("myEnumParamDef" enum "enumChoice2" ("enumChoice1" "enumChoice2" "enumChoice3" "enumChoice4"))`

  In this example "`myEnumParamDef`" is the name of the parameter, `enum` is the data type, "`enumChoice2`" is the default value, and `("enumChoice1" "enumChoice2" "enumChoice3" "enumChoice4")` is the enum choice available.

Example of parameter definition defined by a SKILL expression:

```
("Bulk Offset" float 0.14 float
"ciGetRule(cadr(ciGetRoutingLayers())
\"minSpacing\"
0.1)*ciGetStructPDKMult('CurrentMirror)")
```

In this example "`Bulk Offset`" is the name of the parameter definition, float is the data type, and "`ciGetRule(cadr(ciGetRoutingLayers()) \"minSpacing\" 0.1)*ciGetStructPDKMult('CurrentMirror)`" is the expression definition the default value for this parameter.

`nil` Returns `nil` if the chosen template does not have parameters in its definitions or the template has not been found.

Example

```
;;Pre-requisite: create a template (see documentation for ciTemplateCreate())
cache = ciGetCellView()

cache~>templates~>name
("myCurrentMirror")

tpl = car(cache~>templates)
tpl~>type
"CurrentMirror"

ciTemplateSortParamDefs("CurrentMirror")
(("Sorted Template definition: \n " "CurrentMirror"
    (("Abut All" boolean 1 boolean "t")
        ("Add Dummies" boolean 0 boolean "nil")
```

```
        ("Add GuardRing" boolean 0 boolean "nil")
        ("Bottom Spacing" float 0.0 float "0.0")
        ("Bulk Layer" enum nil
        ("Metal1" "Metal2" "Metal3" "Metal4" "Metal5"
            "Metal6" "Metal7" "Metal8" "Metal9" "Poly"
        ) enum
        "ciCreateRoutingLayerEnumString(2)"
        )
        ("Left Spacing" float 0.0 float "0.0")
        ("MPP" enum nil
        ("N-Tap" "P-Tap" "PO_M1" "M1_M2" "Metal1_8x_bus") enum
        "strcat(buildString(techGetMPPTemplateNames(ciGetTechFile())))"
        )
        ("Right Spacing" float 0.0 float "0.0")
        ("Shape" enum nil
        ("rectangular" "rectilinear") enum
        "\"rectangular rectilinear\""
        )
        ("Spacing" float 0.0 float "0.0")
        ("Style" enum nil
        ("Auto" "SingleRow" "DoubleRow") enum
        "\"Auto SingleRow DoubleRow\""
        )
        ("Top Spacing" float 0.0 float "0.0")
        ("Type" enum nil
        ("ring" "pane") enum
        "\"ring pane\""
        )
        ("Use Fluid" boolean 0 boolean "nil")
        ("Use Min DRC for Spacing" boolean 0 boolean "nil")
        ("Width" float 0.0 float "0.0")
    )
    )
)
```

Reference

`ciTemplateListParams`, `ciTemplateListParamNames`

ciTemplateUpdateParams

```
ciTemplateUpdateParams(
     u_template_id
     l_parameter_list
     => t / nil
```

Description

This function updates the parameter values of the listed parameters for templates that have one or more parameter definitions.

Note: The default values reset the parameter to the default and the storage for the default value will be deleted. Enumerated values will be reset first, then updated rather than appended.

Arguments

u_template_id	Specifies the template ID whose parameter values you want to update.
l_parameter_list	Specifies the list of parameters to be updated.

Values Returned

`t`	Returns `t` if the parameter values are successfully updated.
`nil`	Returns `nil` if the parameter values not updated.

Example

```
;;Pre-requisite: create a template, see documentation for ciTemplateCreate()

cache = ciGetCellView()
cache~>templates~>name
("myCurrentMirror")

tpl = car(cache~>templates)
tpl~>type
"CurrentMirror"


ciTemplateListParams(tpl t) ;; listing non default valued parameters, should list
nothing since all have the default values at this point.
nil

listOfupdatedParams = list(list("Bulk Offset" 0.12) list("Gate Layer" "Metal2")
list("Footer Include File Name" "../.cadence/myFile.il"))
    (("Bulk Offset" 0.12)
```

```
    ("Gate Layer" "Metal2")
    ("Footer Include File Name" "../.cadence/myFile.il")
)
ciTemplateUpdateParams(tpl listOfupdatedParams) ;; Updating the parameter values
for the 3
;; Note in case where a template callback is executed here, other extra statements
can possibly be output here.
t


ciTemplateListParams(tpl t) ;; listing non default parameters again, this time
should return the 3 updated parameters.
(("Bulk Offset" float 0.12)
    ("Footer Include File Name" string "../.cadence/myFile.il")
    ("Gate Layer" enum "Metal2")
)
```

ciToFloat

```
ciToFloat(
     g_value
     )
     => f_result
```

Description

Returns a floating point number for the passed value. If the value is a string, then it will be evaluated and the result returned.

Arguments

g_value	Specifies a value that should either be a floating point number or a string that evaluates to a floating point number.

Values Returned

f_result	Returns the floating point number for the value passed.

Example

```
ciToFloat("0.1*3.0") => 0.3
ciToFloat(3.1768) => 3.1768
```

ciTransferConstraintsInProgress

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

Description

Returns a Boolean to indicate whether constraint transfer is in progress.

Arguments

None

Values Returned

`t`	Constraint transfer is in progress.
`nil`	Constraint transfer is not in progress.

Example

```
;;Transfer of constraints is in progress
ciTransferConstraintsInProgress()
t
```

ciTypeBindingParameter

```
ciTypeBindingParameter(
     S_typeName
     )
     => S_bindingParam / null_string
```

Description

Returns the name of the registered binding parameter for a constraint type or an empty string if none is registered.

Arguments

S_typeName	Specifies the constraint type name.

Values Returned

S_bindingParam	The binding parameter name with which the constraint type was registered.
null_string	An empty string (`""`) if the constraint type was not registered.

Example

Confirm binding parameter has been registered:

```
ciRegTypeBindingParameter("myType" "myBindingParam")
ciTypeBindingParameter('myType)
=> "myBindingParam"
```

ciTypeDefBaseType

```
ciTypeDefBaseType(
     S_typeName
     )
     => S_baseTypeName
```

Description

Returns the base type of the passed custom constraint type. You can define multiple versions of a custom constraint type in `config.xml` by assigning a unique constraint type name to different constraint types that all share the same base type name. This allows multiple versions of the same base constraint type to exist at the same time. To return the version number of a constraint type, see ciTypeDefVersion.

Note: You can also use this function with the VerifyCB callback to determine if an existing constraint is an older version and, if confirmed, displays a message that the constraint should be migrated. This callback can also be used to disable older constraint type versions. For more details on VerifyCB callback, see Using Validation and Verification Callbacks.

For more details on the UI configuration file `config.xml`, see Customizing Constraint Types Using a Configuration File in the *Virtuoso Unified Custom Constraints Configuration Guide*.

Arguments

S_typeName	The constraint type name. It may be specified as a string or symbol.

Value Returned

S_baseTypeName	The constraint base type name. It may be specified as a string or symbol.

Example

The custom constraint types `matchedDevs` and `matchedDevs_vl` are defined in `config.xml` as follows:

```
<ConstraintType>
   <Name>matchedDevs</Name>
   <GUIName menu="Placement">Matched Devices</GUIName>
```

```
   <Version>2</Version>
   <BaseType>matchedDevs</BaseType>
..

<ConstraintType>
   <Name>matchedDevs_v1</Name>
   <GUIName menu="Placement">Matched Devices</GUIName>
   <Version>1</Version>
   <BaseType>matchedDevs</BaseType>
..
```

The results would be as follows:

```
ciTypeDefBaseType('matchedDevs) => 'matchedDevs
ciTypeDefVersion('matchedDevs) => 2

ciTypeDefBaseType('matchedDevs_v1) => 'matchedDevs
ciTypeDefVersion('matchedDevs) => 1
```

ciTypeDefVersion

```
ciTypeDefVersion(
     S_typeName
     )
     => x_version
```

Description

Returns the version number of the passed constraint type. The default version number is `1`. Custom constraint types defined through `config.xml` can specify the version number of the constraint through a version tag.

Each version has its own unique constraint type name, but all versions have the same base type name. This allows multiple versions of the same base constraint type to exist at the same time. To return the base type of a constraint type, see ciTypeDefBaseType.

Note: You can also use this function with the VerifyCB callback to determine if an existing constraint is an older version and, if confirmed, displays a message that the constraint should be migrated. This callback can also be used to disable older constraint type versions. For more details on VerifyCB callback, see Using Validation and Verification Callbacks.

For more details on the UI configuration file `config.xml`, see Customizing Constraint Types Using a Configuration File in the *Virtuoso Unified Custom Constraints Configuration Guide*.

Arguments

S_typeName	The constraint type name. It may be specified as a string or symbol.

Value Returned

x_version	The integer version number.

Example

```
ciTypeDefVersion('symmetry) => 1
```

The custom constraint types `matchedDevs` and `matchedDevs_v1` are defined in `config.xml` as follows:

```
<ConstraintType>
```

```
   <Name>matchedDevs</Name>
   <GUIName menu="Placement">Matched Devices</GUIName>
   <Version>2</Version>
..

<ConstraintType>
   <Name>matchedDevs_v1</Name>
   <GUIName menu="Placement">Matched Devices</GUIName>
   <Version>1</Version>
..
```

The results would be as follows:

```
ciTypeDefVersion('mathedDevs) => 2
ciTypeDefVersion('mathedDevs_vl) => 1
```

ciTypeHasBindingParameter

```
ciTypeHasBindingParameter(
     S_typeName
     )
     => t / nil
```

Description

Checks if a binding parameter has been registered for the specified constraint type.

Arguments

S_typeName	Specifies the constraint type name.

Values Returned

`t`	A binding parameter exists for the specified constraint type.
`nil`	No binding parameter exists for it.

Example

```
ciTypeHasBindingParameter('myType)
=> nil
```

ciTypeIsType

```
ciTypeIsType(
     S_typeName
     )
     => t / nil
```

Description

Checks if the specified `typeName` is a constraint type name, which may be a built-in type, such as distance, or a user-defined custom constraint type that has been specified in a `config.xml` file.

Arguments

S_typeName	Specifies the constraint type name.

Values Returned

`t`	The specified constraint type is a constraint type.
`nil`	The specified constraint type is not a constraint type.

Example

Check if the specified built-in constraint type name, such as `distance`, is a constraint type name. It may be specified as a string or symbol.

```
ciTypeIsType("distance")
  => t
ciTypeIsType('distance)
  => t
```

User-defined constraint type name, `myType`, may be specified as a string or symbol:

```
ciTypeIsType("myType")
  => t
ciTypeIsType('myType)
  => t
```

Non-constraint type names return `nil`:

```
ciTypeIsType("unknown")
  => nil
```

ciTypeIsUserDefined

```
ciTypeIsUserDefined(
     S_typeName
     )
     => t / nil
```

Description

Checks if the specified `typeName` is a user-defined custom constraint type that is specified in a `config.xml` file.

Arguments

S_typeName	Specifies the constraint type name.

Values Returned

`t`	The specified constraint type is a user-defined constraint type.
`nil`	The specified constraint type is not a user-defined constraint type.

Example

Check if the specified constraint type name, `myType`, is user-defined:

```
ciTypeIsUserDefined('myType)
   => t
```

Specify a built-in constraint type, such as `symmetry`, which is not user-defined.

```
ciTypeIsUserDefined('symmetry)
  => nil
```

List all user-defined constraint types.

```
usertypes = setof(ct ciListTypes() ciTypeIsUserDefined(ct))
```

ciTypeListCon

```
ciTypeListCon(
     u_cache
     t_constraintType
     [ g_includeOutOfContext ]
     )
     => l_designObject / nil
```

Description

Lists all the constraints of a given type for a given cache.

Arguments

u_cache	The constraint cache. See `cache`.
t_constraintType	Set a legal constraint type (string or symbol). See `constraint_type`.
g_includeOutOfContext	Lists out of context constraints if set to `t`. This is an optional argument with default of `nil`.

Value Returned

l_designObject	Returns a list of constraints of the given type in the given cache.
`nil`	Template constraint list not found.

ciUniqueMembers

```
ciUniqueMembers(
     l_memberList
     )
     =>l_mems
```

Description

This utility function returns the unique members from the specified member list. It is typically used for member lists that can contain repeated member names due to mFactor expansion.

Arguments

l_memberList	Specifies a member list in the constraint member format. The member list is a list of sublists where each sublist contains the member name and type, and optionally contains its parameters.

Values Returned

l_mems	Returns alphanumeric sorted member list with member types that has been filtered to only include the unique member names.

Example

```
ciUniqueMembers( list( list("MN3" 'inst list("x" 1)) list("MN1" 'inst list("y" 2))
list("MN4" 'inst list("z" 3)) list("MN1" 'inst list("y" 2)) list("MN1" 'inst
list("y" 2)) list("MN3" 'inst list("x" 1)) list("MN333" 'inst list("a" 333))  )  )
     (("MN1" inst)
     ("MN3" inst)
     ("MN333" inst)
     ("MN4" inst)
)
```

ciUnregisterAssistant

```
ciUnregisterAssistant(
     t_assistantName
     )
     => t / nil
```

Description

Unregisters a *Circuit Prospector* assistant (category) that was previously registered using `ciRegisterAssistant`.

See also `ciRegisterAssistant`.

Arguments

t_assistantName	The name of the assistant to be unregistered.

Value Returned

`t`	Action successful. Category no longer displayed in the list of Circuit Prospector categories.
`nil`	Action failed. Category not removed.

Example

```
ciUnregisterAssistant("Pins")
```

ciUnregisterConstraintEditor

```
ciUnregisterConstraintEditor(
     t_constraintEditorName
     )
     => t / nil
```

Description

Unregisters a previously registered constraint editor. If the constraint editor appears in the Constraint Manager's *Constraint Editors* menu, then running this function removes it from the menu list.

See also ciRegisterConstraintEditor.

Arguments

t_constraintEditorName	The name of the constraint editor to be unregistered.

Value Returned

`t`	Removed the specified constraint editor from the Constraint Manager's *Constraint Editors* menu.
`nil`	Failed to remove the specified constraint editor.

Example

```
ciUnregisterConstraintEditor("Matching (strength)")
```

Removes the constraint editor named *Matching (strength)*.

ciUnregisterNetSuperType

```
ciUnregisterNetSuperType(
     t_superType
     )
     => t / nil
```

Description

Removes a net super-type, leaving its sub-types intact.

Arguments

t_superType	Specifies a net super-type.

Values Returned

`t`	Returns `t` if the net super type was removed successfully.
`nil`	Returns `nil` if `t_superType` was not a registered super-type.

Example

To register some super-types:

```
ciRegisterNetSuperType("Priority" '("High" "Low"))
ciRegisterNetSuperType("Supply" '("Power" "Ground"))
ciGetNetSuperTypes() ; ("Priority" "Supply")
```

To remove `"Supply"`:

```
ciUnregisterNetSuperType("Supply")
ciGetNetSuperTypes() ; ("Priority")
```

Reference

ciRegisterNetSuperType

ciUnRegisterTerm

```
ciUnregisterTerm(
     t_libName
     t_cellName
     t_viewName
     t_termName
     )
     => t / nil
```

Description

Unregisters the terminal of a l/c/v if that terminal has already been registered with a default net.

Arguments

t_libName	The library that contains the terminal to be removed.
t_cellName	The cell that contains the terminal to be removed.
t_viewName	The view that contains the terminal to be removed.
t_termName	The terminal to be removed.

Value Returned

`t`	Terminal successfully registered.
`nil`	Action failed. Terminal not unregistered.

Example

```
ciUnRegisterTerm('('("analogLib" "pmos" "symbol")) "D")
```

ciUpdateHierarchicalNotes

```
ciUpdateHierarchicalNotes(
     t_libName
     t_cellName
     t_viewName
     [ g_hierarchical ]
     )
     => t / nil
```

Description

Updates the existing notes on the templates and constraints for a single cellview or all along the hierarchy beginning from the given cellview.

See also ciAddHierarchicalNotes and ciRemoveHierarchicalNotes.

Arguments

t_libName	The library that contains the cellview for which notes are to be updated.
t_cellName	The cell that contains the view for which notes are to be updated.
t_viewName	The view for which notes are to be updated.
g_hierarchical	Determines whether the notes should be updated in the whole hierarchy for the specified cellview. The valid values are `t` to update the notes all along the hierarchy and `nil` to update the notes only to the specified cellview.

Value Returned

`t`	Notes successfully updated for the given cellview.
`nil`	Notes could not be updated.

Examples

To update notes in the entire hierarchy starting from the specified cellview:

```
ciUpdateHierarchicalNotes("myLib" "myCellName" "myView" t)
```

ciUprevEAConstrs

```
ciUprevEAConstrs(
     d_cv
     [ u_cache ]
     )
     => t / nil
```

Description

Modifies and corrects those constraints in the 610EA (early adopter) release which became obsolete in the full 610 release. Specifically, this command will update any layout structure constraints to be modgen, cluster, or cluster boundary constraints.

Arguments

d_cv	The cellview containing the constraints.
u_cache	The constraint specified in the given constraint `cache`.

Value Returned

`t`	Successfully corrected constraints.
`nil`	Constraint correction failed.

Example

```
ciUprevEAConstrs(cv)
```

Performs an update on the constraint in the given cellview.

```
ciUprevEAConstrs(cv cache)
```

Performs an update on the constraint specified in the given constraint cache.

ciUtilsAddNTimes

```
ciUtilsAddNTimes(
     g_val
     x_n
     t_valFmt
     g_addQuotes
     t_sep
     )
     => x_str
```

Description

This utility function is used in modgen constraint generation for creating a string with repeated values.

Arguments

g_val	The value to be added multiple times to the string.
x_n	The number of times the value is to be added.
t_valFmt	The string format of the value.
g_addQuotes	A Boolean to control whether double quotes are added around each value in the string or not.
t_sep	The separator string to be used between each repeated value in the string.

Value Returned

x_str	A string containing the passed value repeated the required number of times with double quotes and separators as specified.

Examples

```
ciUtilsAddNTimes("f"  3 "%s" t " ")
=> "\"f\" \"f\" \"f\" "
```

```
ciUtilsAddNTimes(0.45 2 "%f" nil ",")
=> "0.45,0.45
```

ciUtilsAddQuotes

```
ciUtilsAddQuotes(
     l_stringList
     )
     => t_string
```

Description

This function is used in constraint generation for converting a list of strings into a string where each string in the list will have double quotes when added to the string.

Arguments

l_stringList	A list of strings.

Value Returned

t_string	A single string containing the strings in the passed list of strings where each string has double quotes.

Example

```
ciUtilsAddQuotes('("a" "b" "c"))
=> "\"a\" \"b\" \"c\""
```

ciUtilsBuildString

```
ciUtilsBuildString(
     l_vals
     t_valFmt
     t_sep
     )
     => l_string
```

Description

This is a utility function for building a string from a list of values. The values are converted to strings by applying the passed string format and separated by the passed separator.

Arguments

l_vals	A list of values to be added to a string.
t_valFmt	The string format of the values in the list.
t_sep	The separator to use to separate the values in the string.

Value Returned

l_string	A string containing all the values in the passed list separated by the passed separator.

Example

```
ciUtilsBuildString('(1.1 2.2 3.3) "%f" " ")
=> "1.1 2.2 2.3"
```

ciUtilsMakeUnique

```
ciUtilsMakeUnique(
     l_objLst
     )
     => l_objList
```

Description

This function takes a list of items, which can contain duplicate items and returns a list without any duplicate items.

Arguments

l_objLst	A list of items which can contain duplicates.

Value Returned

l_objList	A list of items which does not contain duplicates.

Examples

```
ciUtilsMakeUnique('(1 2 2 2 1 3 4 5 3 3))
=> list(1 2 3 4 5)
```

```
ciUtilsMakeUnique('("a" "a" "b" "c" "a" "a" "b" "c"))
=> ("a" "b" "c")
```

ciUtilsRemoveNils

```
ciUtilsRemoveNils(
     l_vals
     )
     => l_vals
```

Description

Removes the `nil` elements from a list.

Arguments

l_vals	A list of elements, some of which may be nil.

Value Returned

l_vals	A list of elements, none of which are nil.

Example

```
ciUtilsRemoveNils(list(nil nil 1 2 3 nil nil 4 5 6 nil nil))
=> (1 2 3 4 5 6)
```

ciUtilsRepeatNTimes

```
ciUtilsRepeatNTimes(
     g_val
     x_n
     )
     => l_vals
```

Description

This is a utility function for creating a list containing a single value repeated the specified number of times.

Arguments

g_val	The value to be repeated in the list.
x_n	The umber of times the value should be repeated.

Value Returned

l_vals	A list containing a single value repeated the specified number of times.

Examples

```
ciUtilsRepeatNTimes(0.55 4)
=> '(0.55 0.55 0.55 0.55)
```

```
ciUtilsRepeatNTimes("Metal1" 2)
=> '("Metal1" "Metal1")
```

ciUtilsReplaceNils

```
ciUtilsReplaceNils(
     l_items
     )
     => l_res
```

Description

This is a utility function for replacing any `nil` elements in a list with the specified value. See also, `ciUtilsRemoveNils` and `ciUtilsMakeUnique`.

Arguments

l_items	The list which may contain nil elements.

Value Returned

l_res	A list which does not contain nil elements.

Example

```
ciUtilsReplaceNils(list(nil nil 1 2 3 nil nil 4 5 6 nil nil) 100)
=> (100 100 1 2 3 100 100 4 5 6 100 100)
```

ciWithinConstraint

```
ciWithinConstraint(
     s_conType
     g_cache
     l_objs
     )
     => l_cons
```

Description

Returns the constraints that contain the specified objects as members of the given constraint type.

Arguments

s_conType	The type of constraints to search for.
g_cache	The constraints cache.
l_objs	The list of objects where each object is a sub-list containing the object name and type.

Values Returned

l_cons	A list of constraints of the given type that have the specified objects as its constraint members.

Example

```
ciWithinConstraint('alignment cache '(("NM9" inst)("PM6" inst)))
(ci:0x27ce1630 ci:0x27ea9c70)
```

Circuit Prospector Assistant Customization SKILL Commands

The following SKILL commands are available for customizing the *Circuit Prospector* assistant:

ciA...	
ciActiveSameCellAndSizeIterator	ciAPRCascodeIterator
ciAlignPinsOnCellSide	ciAPRXYInstSymmetricIterator
ciB...	
ciBasicGetParamValue	ciBuildModgenParams
ciBlockResistorArrayIterator	ciBundleSignalsIterator
ciC...	
ciCanCGBeUsed	ciCommonGateAndSourceIterator
ciCascodeSeriesCurrentMirrorIterator	ciCommonGateIterator
ciCategoryListFinderNames	ciCommonSourceIterator
ciClearNetSuperTypes	ciCPRegistrationFromLAM
ciClusterBoundaryForCluster	ciCreateRoutePriorityCon
ciD...	
ciDeviceInfoGetRegisteredParams	ciDeviceInfoRegistry
ciDeviceInfoGetRegisteredTerminals	ciDeviceInfoRestoreDefaultParamNames
ciDeviceInfoRegisterParams	ciDeviceInfoRestoreDefaultTerminalNames
ciDeviceInfoRegisterTerminals	ciDeviceInfoTerminalsAreValid
ciE...	
ciEnableAssistant	ciExpandIteratedDeviceInfo
ciEvaluateGeneratorArgs	
ciF...	
ciFindObjectInHier	
ciG...	
ciGenerateConstraintGroup	ciGetLAMComponentTypes

ciGeneratorCheckInstsNetsPinsInstTerms	ciGetMappedDeviceNames
ciGeneratorForInstSymmetry	ciGetNetNames
ciGeneratorForNetSymmetry	ciGetNetSubTypes
ciGetAction	ciGetNetSuperTypes
ciGetConstraintGroupsEnum	ciGetParamMapping
ciGetCPSelectedResults	ciGetParamName
ciGetDeviceBulkTermName	ciGetParamValue
ciGetDeviceInfo	ciGetParamValueOrDefault
ciGetDeviceNames	ciGetParamValues
ciGetDeviceTermName	ciGetStructure
ciGetFinder	ciGetTechFile
ciGetFirstDeviceTermName	ciGetTechMPPNames
ciGetFluidGuardRingDeviceEnum	ciGetTermNames
ciGetGenerator	ciGuardRingForCluster
ciGetIterator	ciGuardRingForModgen
ciH...	
ciHaveSameParamValues	ciHierarchicalSeriesIterator
ciI...	
ciIgnoreDevice	ciInstsNetsPinsFromSelSet
ciInstGetSplitFingers	ciInstTermIterator
ciInstIterator	ciIsDevice
ciInstListSplitFingers	ciIsNet
ciInstSetSplitFingers	
ciL...	
ciListAllCategoryNames	ciListAllIteratorNames
ciListAllFinderNames	ciListAllStructureNames
ciListAllGeneratorNames	ciListGeneratableConstraintGroups
ciM...	
ciMakeHierContext	ciMOSCascodedCurrentMirrorStructIterator

ciMakeObjectInfo	ciMOSCascodedCurrentMirrorStructIterator2
ciMapParam	ciMOSCascodeIterator
ciMapTerm	ciMOSCommonGateStructIterator
ciMatchedFingerWidth	ciMOSCrossCoupledDifferentialPairStructIterator
ciMatchedParametersForCurrent_Mirror	ciMOSCrossCoupledQuadStructIterator
ciMatchedParamsForInstanceSymmetry	ciMOSCurrentMirrorStructIterator
ciMatchedParamsForSameSizeInstances	ciMOSDifferentialPairStructIterator
ciMergeParams	ciMOSInverterStructIterator
ciModgenForSameCellSizeAndBulk	ciMOSParallelStructIterator
ciMOSActiveLoadStructIterator	ciMOSTransmissionGateStructIterator
ciN...	
ciNetIterator	ciNextConName
ciNetNames	ciNextObjName
ciNetOnTerm	ciNextTemplateName
ciNetPredicates	ciNumTermsEQ2
ciNetRegexs	
ciO...	
ciOrientationForModgen	
ciP...	
ciParallelNetResistorArrayIterator	ciPrintMappedDeviceNames
ciParallelResistorArrayIterator	ciPrintMappedNetNames
ciPinIterator	ciPrintMappedParams
ciPlacerControlledWellGeneration	ciPrintMappedTerminals
ciPrintMappedDefaultNetNames	
ciR...	
ciRegisterAction	ciRegisterNetNames
ciRegisterAssistant	ciRegisterNetPredicate
ciRegisterConstraintGenerator	ciRegisterNetRegexs

ciRegisterDefaultNetName	ciRegisterStructure
ciRegisterDevice	ciResolveNet
ciRegisterDevicesForPDKCategory	ciRunFinder
ciRegisterDynamicParamDef	ciRunFindersAndGenerators
ciRegisterFinder	ciRunGenerator
ciRegisterIterator	ciRunPrecondition
ciRegisterNet	
ciS...	
ciSameCellIterator	ciSetDefaultConstraintEditor
ciSeparateInstsNetsPins	ciSetStructArgVal
ciSeriesResistorArrayIterator	ciSignalIterator
ciSetCMCGSKILLCallbacks	
ciU...	
ciUnexpandDeviceInfo	ciUnregisterConstraintGenerator
ciUnexpandIteratedDeviceInfo	ciUnregisterIterator
ciV...	
ciVariantInfoForFingersAndFingerWidth	
ciX...	
ciXYInstSymmetricIterator	ciXYSortInsts
ciXYNetSymmetricIterator	ciXYSymmetricIterator
ciXYPinSymmetricIterator	

ciActiveSameCellAndSizeIterator

```
ciActiveSameCellAndSizeIterator(
     d_cellviewID
     t_matchExpression
     )
     => l_devices
```

Description

(ICADVM20.1 EXL Only) Returns a list of devices where each sub-list corresponds to two or more active devices that have the same cell and size in the cellview. If the finder match net expression evaluates to `nil`, the device is ignored.

Arguments

d_cellviewID	The cellview to apply the iterator to.
t_matchExpression	The expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_devices	List of devices that match the device in schematic.

Example

```
finder = ciGetFinder("APR Active Same Cell and Size")
insts = ciActiveSameCellAndSizeIterator(geGetEditCellView() finder->expression)
("/I1" "/I2")
```

ciAlignPinsOnCellSide

```
ciAlignPinsOnCellSide(
     u_cache
     l_memberList
     [ ?side t_side ]
     )
     => t / nil
```

Description

Generates alignment constraints for pins on the same side.

Arguments

u_cache	The constraint `cache` for the current cellview.
l_memberList	A single list of instances, pins, instance terminals, and net members that have been selected from the *Circuit Prospector* assistant.
`?side` *t_side*	The side of the pin (top, bottom, left, or right).

Value Returned

`t`	Successfully created new alignment constraint.
`nil`	`Action failed.`

Example

```
ciAlignPinsOnCellSide( ciCacheGet(geGetEditCellView()) list( list("/I1/x" pin)
list("/I2/x" pin)) "top")
```

ciAPRCascodeIterator

```
ciAPRCascodeIterator(
     d_cellViewID
     t_matchExpression
     )
     => l_structuredObjects
```

Description

(ICADVM20.1 Only) Iterator for advanced place and route cascoded structures.

Arguments

d_cellviewID	Database ID of the cellview to which the iterator is to be applied.
t_matchExpression	Expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_structuredObjects	List of objects that satisfy the specified expression.

Example

```
cv = geGetEditCellView()
ciAPRCascodeIterator(cv "t")
```

ciAPRXYInstSymmetricIterator

```
ciAPRXYInstSymmetricIterator(
     d_cellview
     t_finderExpr
     [ ?trigger g_trigger ]
     [ ?likeSchemaSym g_likeSchemaSym ]
     )
     => list / nil
```

Description

(ICADVM20.1 EXL Only) Evaluates the `finderExpr` with the current symmetric pair of objects with common source or drain terminal names that are assigned to `L` and `R` local variables. This iterator is customized for the Auto-Device Placement and Routing flow and is used by the *Circuit Prospector* ADA finders to iterate over all symmetric design instance pairs with common source or drain terminal names, collecting them into symmetric pairs if the result of evaluating the passed expression (`finderExpr`) is not `nil`.

The `L` and `R` variables can be referenced in the `finderExpr`.

In the current schematic, the instance symmetry iterator first looks for symmetrical triggering pairs and then propagates the symmetries from these pairs along the nets and devices that are connected to the triggering pairs and symmetrical pairs which can be one of the following:

- A differential pair made of `fet` and `bjt` devices.
- A pair of instances with the same cell name and same size as `fet` or `bjt` devices with mirrored orientation, aligned on the same Y co-ordinate.

The symmetries are propagated through the nets using terminal names defined for each active device. Symmetries are also transmitted to and propagated through passive devices.

To be a symmetrical pair, both instances must have the same cell name and size.

Symmetries for instances are converted to self-symmetries when there is only one member on the path for symmetry propagation.

Arguments

d_cellview — The cellview that *Circuit Prospector* finders are to iterate over all design instance pairs.

t_finderExpr — The finder expression to be used.

?trigger *g_trigger*

A trigger to capture the symmetries. If set to `t`, the default, the found differential pairs are used to trigger the capture of the symmetries.

If set to `nil`, the differential pairs are not used to trigger the capture of symmetries. Only the pairs of active devices with mirrored orientation, aligned on the same Y coordinate, and without connection to a power supply, are used to trigger the capture of symmetries.

?likeSchemaSym *g_likeSchemaSym*

Sets the orientation of the symmetries triggered by the mirrored active devices.

When set to `t`, the default, the order of the symmetry members is the same as on the schematic. That is, the first member of each matching symmetrical pair should be on the left of the symmetrical axis.

When set to `nil`, the order of the symmetry members is reversed.

Value Returned

`list` — List of symmetric pairs, for example:

```
list(
     list( netA1 netA2 ;;; symmetric pair A
     list( netB1 netB2 ;;; symmetric pair B
     ...
     )
```

`nil` — No instance pairs with common source or drain terminal names were found.

Example

The symmetric pairs with common source or drain terminal names that are assigned to L and R local variables are evaluated and the symmetric pair `MN16` and `MN5` is returned:

```
finderExpr="(L->libName == R->libName) && (L->cellName == R->cellName) &&
     (L->w == R->w) && (L->l == R->l) && (L->r == R->r) && (L->c == R->c)"
symmPairs  = ciAPRXYInstSymmetricIterator(cv finderExpr)
symmPair1  = car(symmPairs)
print(symmPair1~>name)
("MN16" "MN5")
```

ciBasicGetParamValue

```
ciBasicGetParamValue(
     d_deviceId
     t_paramName
     )
     => paramValue / nil
```

Description

This API is similar to ciGetParamValue, except it does not evaluate `iPar` or `pPar` expressions. It gets the parameter value of the user defined parameter. The user-defined parameter was mapped to its PDK names by the ciMapParam function.

Arguments

d_deviceId	The device database ID.
t_paramName	The placeholder name that is used to map the parameter name. The user-defined parameter was mapped to its PDK names by ciMapParam.

Value Returned

paramValue	The value of the mapped parameter for the device.
`nil`	Command failed.

Example

If a mapping for `fetLength` exists `ciMapParam("fetLength" '("l"))` device is set by

`device=car(geGetSelSet())` from the cell view

```
ciBasicGetParamValue(device "fetLength")
```

=> "`340.0n`" ; returns the `fetlength` for that device

ciBlockResistorArrayIterator

```
ciBlockResistorArrayIterator(
     d_cellview
     t_matchExpr
     )
     => l_returnedInsts / nil
```

Description

Iterates over all resistor devices in the cellview based on the result of evaluating the passed or matched expression. This function is used by the *Block Resistor Array* finder of the Circuit Prospector assistant.

Arguments

d_cellview	The cellview containing the design instances to be iterated over.
t_matchExpr	The matched expression string to be used in the iteration.

Value Returned

l_returnedInsts	A list of returned instances. For example: `list(list( inst1 inst2 inst3 ...))`
nil	The device was ignored.

Example

For a block resistor array,

```
matchExpr = "ciIsDevice(device \"resistor\")"
resDevices = ciBlockResistorArrayIterator(geGetEditCellView() matchExpr)
print(resDevices~>name)
```

ciBuildModgenParams

```
ciBuildModgenParams(
     g_cache
     l_devInfo
     l_pattern
     r_nil
     l_args
     )
     => l_modgenParamsAndMembers
```

Description

Creates modgen parameter and member list. This is a utility function.

Arguments

g_cache	The current constraints cache.
s_structType	The type of modgen structure, such as 'DiffPair or 'CurrentMirror.
l_devInfo	The disembodied property list returned by `ciCollectDeviceInfo()`, which contains information about the devices in the modgen.
l_pattern	A list of lists representing the modgen member pattern where each sublist represents a row in the modgen and contains device names for that row and optionally device parameters.
r_modgenRouting	This is reserved for future use.
l_args	The constraint generator arguments that were used to generate the pattern information, where the settings `args->"Add Dummies"`, `args->"Add GuardRing"`, and `args->"Abut All"` will be taken into account, when the modgen parameters and member lists are generated.

Value Returned

l_modgenParamsAndMembers	Creates modgen parameter and member list.

Example

```
devInfo = ciCollectDeviceInfo(cache '(("NM0" inst) ("NM1" inst)) )

pattern = '( ("NM0" "NM1" "NM1" "NM0")
             ("NM1" "NM0" "NM0" "NM1"))
args = args = list( nil stringToSymbol("Add GuardRing") t stringToSymbol("Add
Dummies") nil stringToSymbol("Abut All") t)
modgenParmasAndMembers = ciBuildModgenParams(cache 'diffPair devInfo pattern
modgenRouting args)
modgenParams = car(modgenParmasAndMembers)
(("numRows" 2)
("numCols" 4)
("pattern" "custom ( ( A  B  B  A  )  ( B  A  A  B  ) ) ( ( MX  MX  MX  MX  )  (
R0  R0  R0  R0  ) )")

modgenMembers = cadr(modgenParmasAndMembers)
(("NM0" inst
    (("row" 0)
        ("col" 0)
        ("abutment" 1)
    )
    )
    ("NM1" inst
    (("row" 0)
        ("col" 1)
        ("abutment" 1)
    )
    )
    ("NM1" inst
    (("row" 0)
        ("col" 2)
        ("abutment" 1)
    )
    )
    ("NM0" inst
    (("row" 0)
        ("col" 3)
        ("abutment" 1)
    )
    )
    ("NM1" inst
    (("row" 1)
        ("col" 0)
```

```
        ("abutment" 1)
    )
    )
    ("NM0" inst
    (("row" 1)
        ("col" 1)
        ("abutment" 1)
    )
    )
    ("NM0" inst
    (("row" 1)
        ("col" 2)
        ("abutment" 1)
    )
    )
    ("NM1" inst
    (("row" 1)
        ("col" 3)
        ("abutment" 1)
    )
    )
)
```

ciBundleSignalsIterator

```
ciBundleSignalsIterator(
     d_cellView
     t_matchExpression
     )
     => l_signals / nil
```

Description

Used by the *Circuit Prospector* assistant to iterate over all named bundle and bus nets. It returns the list of corresponding signals. The signals are grouped according to the result of the match expression. If the match expression evaluates to `nil` the signal list is ignored.

Arguments

d_cellView	Design cellview that contains the bundle and bus nets to be iterated.
t_matchExpression	A match expression to group the results together, or spread them into separate bins according to the result. The local variable `bundleSignals`, the value of which is the list of signal names of the current bundle or bus net, can be included in the match expression.

Value Returned

l_signals	The list of signal lists, grouped per the evaluated result of the match expression.
`nil`	`No results found.`

Example

To group all signals from the current schematic cellview that are included into a bundle net called “`data<0:1>,reset,test`”:

```
ciBundleSignalsIterator(geGetEditCellView() “car(bundleSignals)==\”data\””)~>name
=> ((“data<0>” “data<1>” “reset” “test”))
```

ciCanCGBeUsed

```
ciCanCGBeUsed(
     g_ciCon
     t_CGDefName
     g_CG
     )
     => t / nil
```

Description

The default callback used by ciSetCMCGSKILLCallbacks when no user-defined function has been specified for determining if a constraint group should be used or not.

Value Returned

`t`	Constraint group can be used.
`nil`	Command failed.

Example

See ciSetCMCGSKILLCallbacks

ciCascodeSeriesCurrentMirrorIterator

```
ciCascodeSeriesCurrentMirrorIterator(
     d_cellviewID
     t_matchExpression
     )
     => l_devices
```

Description

(ICADVM20.1 EXL Only) Returns a list of corresponding devices that match cascode series current mirror structures in the cellview. If the finder match net expression evaluates to `nil`, the structures are ignored.

Arguments

d_cellviewID	The cellview to apply the iterator to.
t_matchExpression	The expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_devices	List of devices that match the structure in schematic.

Example

```
finder = ciGetFinder("APR Cascode Series Current Mirror")
insts = ciCascodeSeriesCurrentMirrorIterator(geGetEditCellView() finder-
>expression) ("/I1" "/I2")
```

ciCategoryListFinderNames

```
ciCategoryListFinderNames(
     t_categoryName
     )
     => l_finderNames / nil
```

Description

Lists all the finder names for a given category in Circuit Prospector.

Arguments

t_categoryName	Name of the category.

Value Returned

l_finderNames	List of finder names.
nil	The specified category name is invalid. Therefore, no corresponding finders could be found.

Example

```
ciCategoryListFinderNames("Rapid Analog Prototype")
```

The command illustrated above returns the following:

```
("MOS Cascode" "MOS Cascoded Current Mirror" "MOS Cascoded Current Mirror2" "MOS
Current Mirror" "MOS Differential Pair" "MOS Differential Pair - Cross Coupled"
"Passive Arrays" "Active Same Cell large mfactor" "Symmetric Instance Pairs - By
Connectivity" "Capacitor Cluster" "Vertical Orientation" "Negative Supply"
"Positive Supply" "Nets (Symmetry By Connectivity)" "Pins (Symmetry By
Connectivity)" "Top Pins (Alignment)" "Bottom Pins (Alignment)" "Left Pins
(Alignment)" "Right Pins (Alignment)" "Enforce Precedence")
```

ciClearNetSuperTypes

```
ciClearNetSuperTypes(
     )
     => t
```

Description

Clears all registered net super-types.

Arguments

None

Value Returned

`t`	Clears all registered net super-types.

Example

```
;; Register some super-types.
ciRegisterNetSuperType("Priority" '("High" "Low"))
ciRegisterNetSuperType("Supply" '("Power" "Ground"))
ciGetNetSuperTypes() ; ("Priority" "Supply")

;; Clear net super-types.
ciClearNetSuperTypes()
=> t
```

As shown in the above example, all the registered net super-types are cleared.

ciClusterBoundaryForCluster

```
ciClusterBoundaryForCluster(
     u_cache
     l_instances
     )
     => l_clusterBoundary / nil
```

Description

Used by the *Circuit Prospector* assistant as a generator to set a *Cluster Boundary* constraint to an existing *Cluster* constraint. The *flexibleFlag* parameter of the *Cluster Boundary* is set to `1` (`true`).

Arguments

u_cache	Constraint cache that contains the *Cluster* constraints to be iterated, and in which the *Cluster Boundary* constraints must be created.
l_instances	List of members of the same *Cluster* constraint that are selected in the *Circuit Prospector* assistant's browser.

Value Returned

l_clusterBoundary	A list made of the constraint ID for the new *Cluster Boundary* constraint when it is created.
`nil`	`No actions completed.`

Example

If "`NM1`" and "`NM2`" are the names of two instances that are members of an existing *Cluster* constraint, the following functions creates a *Cluster Boundary* and the member of that new *Cluster Boundary* is the *Cluster* constraint for "`NM1`" and "`NM2`":

```
cache = ciCacheGet(geGetEditCellView()
ciClusterBoundaryForCluster(cache list( list("/NM1" inst) list("/NM2" inst)))
```

ciCommonGateAndSourceIterator

```
ciCommonGateAndSourceIterator(
     d_cellviewID
     t_matchExpression
     )
     => l_devices
```

Description

(ICADVM20.1 EXL Only) Returns a list of corresponding devices that match the common gate and source structures in the cellview. If the finder match net expression evaluates to `nil`, the structures are ignored.

Arguments

d_cellviewID	The cellview to apply the iterator to.
t_matchExpression	The expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_devices	List of devices that match the structure in schematic.

Example

```
finder = ciGetFinder("APR Active Same Size Common Gate and Source")
cv= geGetEditCellView
ciCommonGateAndSourceIterator(cv "t")
```

Returns the common gate and source structures that match the expression

```
ciCommonGateAndSourceIterator(geGetEditCellView() finder->expression) ("/I1" "/
I2")
```

ciCommonGateIterator

```
ciCommonGateIterator(
     d_cellviewID
     t_matchExpression
     )
     => l_devices
```

Description

(ICADVM20.1 EXL Only) Returns a list of corresponding devices that match the common gate structures in the cellview. If the finder match net expression evaluates to `nil`, the structures are ignored.

Arguments

d_cellviewID	The cellview to apply the iterator to.
t_matchExpression	The expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_devices	List of devices that match the structure in schematic.

Example

```
finder = ciGetFinder("APR Active Same Size Common Gate")
insts = ciCommonGateIterator(geGetEditCellView() finder->expression)
```

ciCommonSourceIterator

```
ciCommonSourceIterator(
     d_cellviewID
     t_matchExpression
     )
     => l_devices
```

Description

(ICADVM20.1 EXL Only) Returns a list of corresponding devices that match the common source structures in the cellview. If the finder match net expression evaluates to `nil`, the structures are ignored.

Arguments

d_cellviewID	The cellview to apply the iterator to.
t_matchExpression	The expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_devices	List of devices that match the structure in schematic.

Example

```
finder = ciGetFinder("APR Active Same Size Common Source")
insts = ciCommonSourceIterator(geGetEditCellView() finder->expression) ("/I1" "/
I2")
```

ciCPRegistrationFromLAM

```
ciCPRegistrationFromLAM(
     l_componentTypes
     [ ?addNewLines g_addNewLines ]
     )
     => t_cpRegistrationString
```

Description

Used in conjunction with ciGetLAMComponentTypes to create a string containing `ci` registration function calls to register the device, terminal, and parameter mappings specified in a `cph.lam` file associated with a particular PDK. The device and terminal registrations are needed for the Circuit Prospector finders, iterators, and generators to function with the selected PDK. The returned string can either be evaluated using `evalstring()` to call the `ci` registration functions, or the string can be printed out using `printf` so that the `ci` registration calls can be placed in a file loaded by the `libInit.il` file of the PDK.

Arguments

l_componentTypes	The component type list returned by calling the `ciGetLAMComponentTypes` function.
`?addNewLines` *g_addNewLines*	The optional Boolean value that defaults to `nil` for adding new lines into the returned string. Set this argument to `t` if the string needs to be printed to a file. Otherwise, set it to `nil` if the string needs to be evaluated with `evalstring().`

Value Returned

t_cpRegistrationString	Returns a string containing the `ci` registration function calls to register the device, terminal, and parameter name mappings specified in the component type list. **Note:** If there is any problem in running the SKILL function, an empty string (`""`) is returned and warning is displayed.

Example

```
gpdk045_compTypes = ciGetLAMComponentTypes(geGetEditCellView() "gpdk045")
info(ciCPRegistrationFromLAM(gpdk045_compTypes ?addNewLines t))

 ciRegisterDevice("nfet" append(ciGetDeviceNames("nfet") '(
 ("gpdk045" "nmos1v" nil)
 ("gpdk045" "nmos1v_3" nil)
)))
 ciRegisterDevice("pfet" append(ciGetDeviceNames("pfet") '(
 ("gpdk045" "pmos1v" nil)
 ("gpdk045" "pmos1v_3" nil)
)))
 ciRegisterDevice("fet" append(ciGetDeviceNames("fet") '(
 ("gpdk045" "nmos1v" nil)
 ("gpdk045" "nmos1v_3" nil)
)))
 ciRegisterDevice("fet" append(ciGetDeviceNames("fet") '(
 ("gpdk045" "pmos1v" nil)
 ("gpdk045" "pmos1v_3" nil)
)))
 ciMapParam("fingerWidth" append(ciGetParamMapping("fingerWidth") '("fw")))
 ciMapParam("lxActiveLayer" append(ciGetParamMapping("lxActiveLayer") '("Oxide
drawing")))
 ciMapParam("lxMaxWidth" append(ciGetParamMapping("lxMaxWidth") '("1e-06")))
```

ciCreateRoutePriorityCon

```
ciCreateRoutePriorityCon(
     u_cache
     l_instsNetsPins
     v_value
     )
     => ciCon / nil
```

Description

Used by *Circuit Prospector* constraint generators to generate routing priority constraints.

Arguments

u_cache	The constraint cache (see cache).
l_instsNetsPins	List of database objects.
v_value	The routing priority to be set.

Value Returned

ciCon	Routing priority constraint successfully generated.
nil	Command failed.

Example

```
routePriorityCon = ciCreateRoutePriorityCon(cache instsNetsPins 5)
```

ciDeviceInfoGetRegisteredParams

```
ciDeviceInfoGetRegisteredParams(
     [ t_deviceTypeName ]
     )
     => l_paramNames
```

Description

Returns the parameters registered for the specified device type.

Arguments

t_deviceTypeName	The name of the device type. If the `deviceTypeName == 'default`, the `ciDeviceInfoGetRegisteredParams` SKILL function returns the default parameters that can be used by other SKILL functions, such as ciCollectDeviceInfo.

Value Returned

l_paramNames	An association list that maps the parameters to their default values for the given device type, such as `mos`, `bjt`, and so on.

Example

- Returns a default parameters list:

```
ciDeviceInfoGetRegisteredParams()
(("mFactor" 1)
 ("fingerCount" 1)
 ("length" nil)
 ("width" nil)
 ("fingerWidth" nil)
)
```

- Returns the default parameters list because no paremeters were registered for `resistor`:

```
ciDeviceInfoGetRegisteredParams("resistor")
(("mFactor" 1)
```

```
 ("fingerCount" 1)
 ("length" nil)
 ("width" nil)
 ("fingerWidth" nil)
)
```

- When you register parameters for `resistor` using the ciDeviceInfoRegisterParams SKILL function as following:

  ```
  ciDeviceInfoRegisterParams("resistor" list('("width" 1) '("length" 1)
                             '("area" 1)))
  t
  ```

  The `ciDeviceInfoGetRegisteredParams` SKILL function returns the list of newly registered parameters for `resistor`:

  ```
  ciDeviceInfoGetRegisteredParams("resistor")
  (("width" 1)
   ("length" 1)
   ("area" 1)
  )
  ```

ciDeviceInfoGetRegisteredTerminals

```
ciDeviceInfoGetRegisteredTerminals(
     [ t_deviceTypeName ]
     )
     => l_terminalNames
```

Description

Returns the terminals registered for the specified device type.

Arguments

t_deviceTypeName	The name of the device type. If the `deviceTypeName == 'default`, the `ciDeviceInfoGetRegisteredTerminals` SKILL function returns the default terminal that can be used by other SKILL functions, such as ciCollectDeviceInfo.

Value Returned

l_terminalNames	The list of terminals registered for the given device type.

Example

- When default terminals have not been overridden, the `ciDeviceInfoRegisterTerminals` SKILL function returns a list of all the registered default terminals:

  ```
  ciDeviceInfoGetRegisteredTerminals()
  ("source" "gate" "drain" "bulk")
  ```

- When you register the different default terminals for `resistor` using the ciDeviceInfoRegisterTerminals SKILL function:

  ```
  ciDeviceInfoRegisterTerminals("resistor" list("minus" "plus"))
  t
  ```

  The `ciDeviceInfoGetRegisteredTerminals` SKILL function returns a list of newly registered default terminals for `resistor`:

  ```
  ciDeviceInfoGetRegisteredTerminals("resistor")
  ```

```
("minus" "plus")
```

ciDeviceInfoRegisterParams

```
ciDeviceInfoRegisterParams(
     { t_deviceTypeName | 'default | nil }
     [ (t_paramName ... t_paramNameN) | 'default | '(nil) ]
     )
     => t / nil
```

Description

Registers or unregisters parameters for a device type, such as `bjt`, `mos`, and so on.

- To register a parameter,
  - Specify a `deviceTypeName` as a string or specify the symbol, `'default`, to register default parameters, that is, parameters for device type that have not been registered using `ciDeviceInfoRegisterParams`. For example:

    ```
    ciDeviceInfoRegisterParams("mos"  '(("mosParam1" "defaultValueForParam1")
    ("mosParam2" nil)))
    t
    ciDeviceInfoRegisterParams(default '(("default1" "defaultValue1")))
    t
    ciDeviceInfoGetRegisteredParams("mos")
    (("mosParam1" "defaultValueForParam1") ("mosParam2" nil))
    ciDeviceInfoGetRegisteredParams("noParamRegistered")
    ("default1" "defaultValue1")
    ```

  - Specify the `paramNames` argument as an association list containing the parameter names and their default values. To specify that there is no parameter for a `deviceTypeName`, set `paramNames` as `'(nil)`. This can be useful for some device types, such as `passive`.
- To unregister all the parameters for all the device types, instead of specifying the `deviceTypeName` argument, specify `nil` as the first argument.
- To unregister parameters for a given device type, do the following:

  a. Set the `deviceTypeName` argument to the device type for which parameters need to be unregistered.

  b. Set the value of the required parameters to `nil`.

Arguments

`t_deviceTypeName | 'default | nil`

Specifies whether to register or unregister parameters for a given device type.

t_deviceTypeName	Specifies the name of the device type for which parameters need to be registered, such as `("bjt", "mos", ...)`.
`'default`	Registers the default parameters for `deviceTypeName` that do not have registered parameters.
`nil`	Specifies to unregister all the parameters for all the device types.

`(`*t_paramName* `...` *t_paramNameN*`) | 'default | '(nil)`

t_paramName	The parameter name and the corresponding value. For example, `list('("param1" defaultVal1) '("param2" defaultVal2) ...)` **Note:** If you specify the value of a `paramName` as `nil`, then that parameter gets unregistered for the specified device type.
`'default`	Registers the default parameters explicitly for the specified device type name.
`'(nil)`	Specifies that no parameters are registered for the specified device type name.

Value Returned

`t`	Successfully registered or unregistered the parameters for the given device type.
`nil`	Command failed.

Example

```
ciDeviceInfoGetRegisteredParams("resistor") ;; returns default parameter list
because no parameters were registered for "resistor"
(("mFactor" 1)
    ("fingerCount" 1)
    ("length" nil)
    ("width" nil)
```

```
    ("fingerWidth" nil)
)

ciDeviceInfoRegisterParams("resistor" list('("width" 1) '("length" 1) '("area"
1))) ;; register parameters for "resistor"
t

ciDeviceInfoGetRegisteredParams("resistor") ;; returns the new parameter list for
"resistor"
(("width" 1)
    ("length" 1)
    ("area" 1)
)
ciDeviceInfoRegisterParams('default '(("length" 1) ("width" 1) ("perimeter" 4)
("area" 1))) ;; overrided default parameter
t
ciDeviceInfoGetRegisteredParams("mos") ;; returns default parameter list because
no parameters were registered for "mos"
(("length" 1)
    ("width" 1)
    ("perimeter" 4)
    ("area" 1)
)

ciDeviceInfoRegisterParams("mos" '(("length" 1) ("width" 1) ("perimeter" 4)
("area" 1) ("mfactor" 1))) ;; register parameter for "mos"
t

ciDeviceInfoGetRegisteredParams("mos") ;; returns the parameter registered for
"mos"
(("length" 1)
    ("width" 1)
    ("perimeter" 4)
    ("area" 1)
    ("mfactor" 1)
)

ciDeviceInfoRegisterParams("mos") ;; Unregister parameters for "mos"
t

ciDeviceInfoGetRegisteredParams("mos") ;; returns the default parameters because
"mos" parameters have been unregistered.
(("length" 1)
```

```
    ("width" 1)
    ("perimeter" 4)
    ("area" 1)
)

ciDeviceInfoGetRegisteredParams("resistor") ;; "resistor" parameter are still
registered.
(("width" 1)
    ("length" 1)
    ("area" 1)
)

ciDeviceInfoRegisterParams(nil) ;; Unregister all the parameters for all the device
types
t

ciDeviceInfoGetRegisteredParams("resistor") ;; Returns the default parameter
because all parameters have been unregistered.
(("length" 1)
    ("width" 1)
    ("perimeter" 4)
    ("area" 1)
)
```

ciDeviceInfoRegisterTerminals

```
ciDeviceInfoRegisterTerminals(
     { t_deviceTypeName | 'default | nil }
     [ (t_terminalName ... t_terminalNameN) | 'default | '(nil) ]
     )
     => t / nil
```

Description

Registers or unregisters terminals for a device type, such as `bjt`, `mos`, and so on.

- To register a terminal,
  - Specify a `deviceTypeName` as a string or specify the symbol, `'default`, to register default terminals, that is, terminals for device type that have not been registered using `ciDeviceInfoRegisterTerminals`..
  - Specify the `terminalName` argument as a string list containing the terminal names.
- To unregister all the terminal for all the device types, instead of specifying the `deviceTypeName` argument, specify `nil` as the first argument.
- To unregister terminals for a given device, do the following:
  a. Set the `deviceTypeName` argument to the device for which terminals need to be unregistered.
  b. Set the required terminals to `nil`.

Arguments

`t_deviceTypeName | 'default | nil`

Specifies whether to register or unregister terminals for a given device type.

t_deviceTypeName	Specifies the name of the device type for which terminals need to be registered, such as `("bjt", "mos", ...)`
`'default`	Registers the default terminals for `deviceTypeName` that do not have any registered terminals.
`nil`	Specifies to unregister all the terminals for all the device types.

`(t_terminalName ... t_terminalNameN) | 'default | '(nil)`

t_terminalName	Specifies a list containing the terminal names, such as, `list("source" "drain" ...)`. **Note:** If you specify `nil` instead of a terminal name, the all terminals for the specified device type get unregistered.
`'default`	Registers the default terminals explicitly for the specified device type.
`'(nil)`	Specifies that no terminals are registered for the specified device type name.

Value Returned

`t`	Successfully registered or unregistered the terminals for the given device type.
`nil`	Command failed.

Example

```
ciDeviceInfoGetRegisteredTerminals("resistor") ;;nothing has been registered, so
returns the default terminal list
("source" "gate" "drain" "bulk")

ciDeviceInfoRegisterTerminals("resistor" '("plus" "minus")) ;;registers terminals
for "resistor"
t

ciDeviceInfoGetRegisteredTerminals("resistor") ;;now, as terminals have been
registered for "resistor", the new set of terminals is returned
("plus" "minus")
ciDeviceInfoRegisterTerminals("opamp" '("in_min" "in_plus" "out"))
t

ciDeviceInfoGetRegisteredTerminals("opamp")
("in_min" "in_plus" "out")

ciDeviceInfoRegisterTerminals("opamp") ;;unregister terminals for "opamp".
        ;;you can also use ciDeviceInfoRegisterTerminals("opamp" nil) instead to
        unregister the "opamp" terminal
t
```

```
ciDeviceInfoGetRegisteredTerminals("opamp") ;;nothing has been registered for
"opamp", so returns the default terminal list
("source" "gate" "drain" "bulk")

ciDeviceInfoGetRegisteredTerminals("resistor") ;;terminals are still registered
for "resistor"
("plus" "minus")

ciDeviceInfoRegisterTerminals(nil) ;;unregister terminals for all device types
t

ciDeviceInfoGetRegisteredTerminals("resistor") ;;terminals for "resistor" have
been unregistered, so the default terminals are returned
("source" "gate" "drain" "bulk")
```

ciDeviceInfoRegistry

```
ciDeviceInfoRegistry(
    )
    => g_deviceInfo
```

Description

Returns the registry where information about all the device types is stored. This registry is used by ciCollectDeviceInfo.

Arguments

None

Value Returned

g_deviceInfo	The table containing information about all the device types.

Example

```
ciDeviceInfoRegistry()
table:deviceInfos
```

ciDeviceInfoRestoreDefaultParamNames

```
ciDeviceInfoRestoreDefaultParamNames(
     )
     => l_defaultParams
```

Description

Restores device parameters to default parameters.

Arguments

None

Value Returned

l_defaultParams	The list of default parameters.

Example

```
ciDeviceInfoRegisterParams('default '(("length" 1) ("width" 1) ("perimeter" 4)
("area" 1))) ;; overrides default parameter
t

ciDeviceInfoGetRegisteredParams() ;; returns the default parameters.
(("length" 1)
 ("width" 1)
 ("perimeter" 4)
 ("area" 1)
)

ciDeviceInfoRestoreDefaultParamNames() ;; restores device parameters to default
parameters.
(("mFactor" 1)
 ("fingerCount" 1)
 ("length" nil)
 ("width" nil)
 ("fingerWidth" nil)
)

ciDeviceInfoGetRegisteredParams() ;; default parameters have been restored
```

```
(("mFactor" 1)
 ("fingerCount" 1)
 ("length" nil)
 ("width" nil)
 ("fingerWidth" nil
)
```

ciDeviceInfoRestoreDefaultTerminalNames

```
ciDeviceInfoRestoreDefaultTerminalNames(
     )
     => l_defaultTerminals
```

Description

Restores device terminals to default terminals.

Arguments

None

Value Returned

l_defaultTerminals	The list of default terminals.

Example

```
ciDeviceInfoGetRegisteredTerminals() ;; returns default terminal names
("source" "gate" "drain" "bulk")

ciDeviceInfoRegisterTerminals('default '("source" "drain" "gate")) ;;overrides
default terminal names
t

ciDeviceInfoGetRegisteredTerminals() ;; returns the new default terminal names
("source" "drain" "gate")

ciDeviceInfoRestoreDefaultTerminalNames() ;; restores the default terminal names
("source" "gate" "drain" "bulk")

ciDeviceInfoGetRegisteredTerminals() ;; returns default terminal names that were
restored
("source" "gate" "drain" "bulk")
```

ciDeviceInfoTerminalsAreValid

```
ciDeviceInfoTerminalsAreValid(
     l_terminalNames
     )
     => t / nil
```

Description

Checks whether the specified device terminals are valid inputs for ciCollectDeviceInfo.

Arguments

l_terminalNames	The list of terminal names registered terminals for a device family, such as `bjt`, `mos`, and so on.

Value Returned

`t`	The specified list of terminals names is valid.
`nil`	Command failed.

Example

```
ciDeviceInfoTerminalsAreValid(list('a))
nil

ciDeviceInfoTerminalsAreValid('("minus" "plus"))
t
```

ciEnableAssistant

```
ciEnableAssistant(
     t_assistantName
     g_enable
     )
     => t / nil
```

Description

Enables or disables a Circuit Prospector assistant (category) so that it is visible/not visible in the list of Circuit Prospector categories.

Arguments

t_assistantName	The name of the assistant (category) to be made visible or hidden.
g_enable	The show hide state, `t` or `nil`, of the category.

Value Returned

`t`	Successfully changed visible state of the given category.
`nil`	Command failed.

Example

```
ciEnableAssistant("Pins" nil)
```

Hides the *Pins* category in the *Circuit Prospector*

```
ciEnableAssistant("Pins" t)
```

Shows the *Pins* category in the *Circuit Prospector*

ciEvaluateGeneratorArgs

```
ciEvaluateGeneratorArgs(
     u_cache
     t_defaultCGen
     l_instNetsPinsTerms
     [ ?triggerCBinMode t_triggerCBinMode ]
     [ ?argValuesToUpdate l_argValuesToUpdate ]
     )
     => l_argumentValues
```

Description

Evaluates the specified constraint generator argument values.

Arguments

u_cache — The constraint `cache`.

t_defaultCGen — The name of the default constraint generator for which arguments need to be evaluated.

l_instNetsPinsTerms — The list of instances, nets, pins, and terminals that need to be passed to evaluate the arguments.

`?triggerCBinMode` *t_triggerCBinMode*

The mode in which the callbacks need to be called when `argValuesToUpdate` are specified. Valid values are `'userEdit` and `'createNewTemplate`.

Note: Any other symbol does not trigger the callbacks for the argument values that need to be updated.

`?argValuesToUpdate` *`l_argValuesToUpdate`*

The list of argument values that need to be updated after the first evaluation of the argument. This list can be one of the following:

- A disembodied property list (DPL) starting with `nil`
- A DPL without the `nil` as the first element of the list
- An association list

Value Returned

`l_argumentValues`

A DPL containing the values of each argument. No widget properties are appended to this DPL.

Example

```
ciEvaluateGeneratorArgs(ciGetCellView() "GenericModgen" list('("M1" inst) '("M2"
inst) '("INN" net)))
(nil Device\ mFactors " M1 (4) M2 (4)" Device\ Mapping " A:M1 B:M2"
Num\ Rows 1 Pattern " A A A A B B B B" Orientation
"R0 R0 R0 R0 R0 R0 R0 R0" Default\ Orientation "R0" Select\ Edge ""
Operation "" Device\ Horizontal\ Spacing 0.0 Device\ Vertical\ Spacing
0.0 Abut\ All t Dummy\ Net "INN"
Dummy\ Parameters "neighbor" Dummy\ Num\ Fingers 0 Dummy\ Length
0.0 Dummy\ Width 0.0 Add\ GuardRing nil
Type "ring" Shape "rectangular" Net
"INN" Spacing 0.0 Left\ Spacing 0.0
Right\ Spacing 0.0 Top\ Spacing 0.0 Bottom\ Spacing
0.0 MPP "N-Tap(gpdk045)" Use\ Fluid nil
Device "undefined" Width 0.0 Use\ Min\ DRC\ for\ Spacing
nil Layer\ Filter "" Merge\ layers ""
Layers\ preset "default"
)

ciEvaluateGeneratorArgs(ciGetCellView() "GenericModgen" list('("M1" inst) '("M2"
inst) '("INN" net)) ?triggerCBinMode 'userEdit ?argValuesToUpdate list(list("Num
Rows" 2)))
(nil Device\ mFactors " M1 (4) M2 (4)" Device\ Mapping " A:M1 B:M2"
```

```
Num\ Rows 2 Pattern "A A A A \nB B B B" Orientation
"R0 R0 R0 R0 \nR0 R0 R0 R0" Default\ Orientation "R0" Select\ Edge ""
Operation "" Device\ Horizontal\ Spacing 0.0 Device\ Vertical\ Spacing
0.0 Abut\ All t Dummy\ Net "INN"
Dummy\ Parameters "neighbor" Dummy\ Num\ Fingers 0 Dummy\ Length
0.0 Dummy\ Width 0.0 Add\ GuardRing nil
Type "ring" Shape "rectangular" Net
"INN" Spacing 0.0 Left\ Spacing 0.0
Right\ Spacing 0.0 Top\ Spacing 0.0 Bottom\ Spacing
0.0 MPP "N-Tap(gpdk045)" Use\ Fluid nil
Device "undefined" Width 0.0 Use\ Min\ DRC\ for\ Spacing
nil Layer\ Filter "" Merge\ layers ""
Layers\ preset "default"
)
```

Note: `?argValuesToUpdate` can be specified as one of the following:

```
list(list("Num Rows" 2))
list(nil "Num Rows" 2)
list("Num Rows" 2)
list('Num\ Rows 2), ....
```

For the argument name, symbols can also be used instead of string.

ciExpandIteratedDeviceInfo

```
ciExpandIteratedDeviceInfo(
     l_deviceInfo
     )
     => l_expandedDeviceInfo
```

Description

Expands any iterated device names in a device information list returned by `ciCollectDeviceInfo`.

Arguments

l_deviceInfo	The device information list returned by calling `ciCollectDeviceInfo`.

Value Returned

l_expandedDeviceInfo	A device information list where the iterated device names have been expanded.

Example

Collect the device information for `MN1<0:2>` and `MN2` while treating `MN1<0:2>` as a single device:

```
devInfo = ciCollectDeviceInfo(cache '(("MN1<0:2>" inst) ("MN2" inst)))
mapcar(lambda((dev) dev->name) devInfo->devs) => '("MN1<0:2>" "MN2")
```

Then, expand any iterated devices out into individual devices, for example, `MN1<0:2> => MN1<0>`, `MN1<1>` and `MN1<2>`, as shown below:

```
expandedDevInfo = ciExpandIteratedDeviceInfo(devInfo)
mapcar(lambda((dev) dev->name) expandedDevInfo->devs) => '("MN1<0>" "MN1<1>"
"MN1<2>" "MN2") ;;; devs is a list of 4 devices
```

ciFindObjectInHier

```
ciFindObjectInHier(
     d_cache
     t_objectFullPathName
     s_objectType
     )
     => dbId / nil
```

Description

Finds the database ID of the specified hierarchical object. This function is used by Circuit Prospector constraint generators.

Arguments

d_cache	Constraint view in which constraints are being created. This is either the current cache or the cache associated with the cellview in which the hierarchical object is contained.
t_objectFullPathName	The path to the object.
s_objectType	The object type, which has one of the following values: `'inst`, `'net`, `'pin`, `'instTerm`, or `'signal`. ■ `'inst`: Returns the instance (for example, `I0`, `I0<10:0>`) or bit instance(for example, `I0<0>`) ■ `'instTerm`: Returns the terminal on an instance or bit instance, if it is found. ■ `'pin`: Returns the physical pin on a terminal. ■ `'net`: Returns the net. When the object type is `'net`, the `ciFindObjectInHier` function also returns a signal if it is found in the hierarchy. **Note:** A leading slash in the specified object name means that you have given an absolute path to the object. If there is no leading slash in the given object name, the function searches the cellview associated to the given cache and below.

Value Returned

dbID	Successfully changed visible state of the given category.
nil	Command failed.

Example

```
subCache = ciCacheGet("amsPLL" "vco2phase" "schematic")

obj = ciFindObjectInHier("/I7/I9/MN1" subCache 'inst)

obj->name
"MN1"

obj->cellView->cellName
"vco2phase"
```

ciGenerateConstraintGroup

```
ciGenerateConstraintGroup(
     g_ciCon
     t_CGDefName
     t_CGName
     )
     => t / nil
```

Description

The default constraint group generation callback used by ciSetCMCGSKILLCallbacks when no user-defined function has been specified.

Value Returned

`t`	Constraint group has been created successfully.
`nil`	Command failed.

Example

See ciSetCMCGSKILLCallbacks.

ciGeneratorCheckInstsNetsPinsInstTerms

```
ciGeneratorCheckInstsNetsPinsInstTerms(
     l_instsNetsPins
     )
     => t / nil
```

Description

Checks whether the constraint generator has been called with the expected number of instances, nets, pins, and instTerms. This function is used within the constraint generators. If a wrong number of instances, nets, pins, or instTerms have been provided then a warning message will be displayed in the CIW.

Arguments

l_instsNetsPins	Specifies the disembodied list of objects.

Value Returned

`t`	Returns the expected number of instances, nets, pins, and instTerms.
`nil`	Command failed.

Example

The following snippets illustrate the use of the `ciGeneratorCheckInstsNetsPinsInstTerms` SKILL command:

Example 1

```
inp = '(("/MPS" inst) ("/MP7" inst) ("/MP" inst) ("/net014" net) ("/VDD" net))
inpDPL = ciSeparateInstsNetsPins(inp)
ciGeneratorCheckInstsNetsPinsInstTerms( "TestGen"  inpDPL 2 2 0 0)
=> t
```

Example 2

```
ciGeneratorCheckInstsNetsPinsInstTerms( "TestGen"  inpDPL 1 2 0 2)
=> nil
```

```
*WARNING* (CMGR-6002): Incorrect number of insts, nets, pins or instTerms to run
Constraint Generator "TestGen". This generator requires 1 inst(s), 2 net(s) 0
pin(s) and 2 instTerm(s).
```

ciGeneratorForInstSymmetry

```
ciGeneratorForInstSymmetry(
     u_cache
     l_insts
     )
     => ciCon / nil
```

Description

Used by the *Circuit Prospector* constraint generators to generate instance symmetry constraints.

Arguments

u_cache	The constraint cache (see cache).
l_insts	List of the instances that are to form part of the instance symmetry constraint.

Value Returned

ciCon	Instance symmetry constraint successfully generated.
nil	Command failed.

Example

```
instSymmetryCon = ciGeneratorForInstSymmetry(cache insts)
```

ciGeneratorForNetSymmetry

```
ciGeneratorForNetSymmetry(
     u_cache
     d_nets
     )
     => ciCon / nil
```

Description

Used by the *Circuit Prospector* constraint generators to generate net symmetry constraints.

Arguments

u_cache	The constraint cache (see cache).
d_nets	The nets that are to form part of the instance symmetry constraint.

Value Returned

ciCon	Net symmetry constraint successfully generated.
nil	Command failed.

Example

```
netSymmetryCon = ciGeneratorForNetSymmetry(cache nets)
```

ciGetAction

```
ciGetAction(
     t_actionName
     )
     => l_actionAttributes
```

Description

Returns a disembodied property list (DPL) of attributes associated with the specified action name. This function is an alias for the `ciGetGenerator` function. The purpose of this alias is to emphasize the fact that constraint generator SKILL expressions can do everything possible in SKILL and not just create constraints.

See also ciRegisterAction, ciRegisterConstraintGenerator, and ciGetGenerator.

Arguments

t_actionName	Name of the action.

Value Returned

l_actionAttributes

A DPL of the action and its attributes in the following format:

```
list(nil 'name <name>
    'description "description"
    'expression "expression"
    ['addToToolbar t]
    ['iconName "icon name"]
    ['args "argExprStr"|list( list("argName"
    'argType argVal1 argVal2 ...)]
    ['menu "menuName"|list("menuName1"
    "menuName2" "menuName3")]
    ['forcePopup t]
    ['preconditon "preconditionExpr"]
    ['callback "callbackExpr"]
    ['useCGenForEdit t]
    ['title "titleName"]
    ['size list(width, height)]
    ['templateName "templateName"]
    ['settings list(nil settingName value
    ...)
```

Example

If the example given for ciRegisterAction is considered, the following:

```
ciGetAction("List Constraints")
```

Returns:

```
list(nil
    'name "List Constraints"
    'description "List the constraints in the CIW"
    'expression "when(args->\"List Types\" printf(\"%L\" cache-
    >constraints~>type)) when(args->\"List Names\" printf(\"%L\" cache-
    >constraints~>name))"
    'addToToolbar t
    'iconName "listConstraints"
    'args list(
        list("List Types" 'bool t)
        list("List Names" 'bool t)
    )
)
```

ciGetConstraintGroupsEnum

```
ciGetConstraintGroupsEnum(
     g_cache
     )
     => l_constraintGroupName
```

Description

Returns a string to be used as an `enum` variable for selecting the default constraint group to be used by the Pin to Trunk router. The result can be used as the value of an argument for a constraint generator. The resulting `enum` variable is the same as the one used by the Pin to Trunk GUI of the Modgen Pattern Editor. Consistent with the Modgen Pattern Editor, it sets the default to `virtuosoDefaultSetup` if that choice is available.

Also see the ciRegisterConstraintGenerator SKILL function for more information on registering constraint generators and related arguments.

Arguments

g_cache	The constraints cache.

Value Returned

l_constraintGroupName	A string containing a list of the constraint group names. **Note:** If there is any problem in running the SKILL function, an empty string (`""`) is returned and warning is displayed.

Example

```
ciGetConstraintGroupsEnum(ciGetCellView())
"6xSpacing 4xSpacing 2xWidth virtuosoDefaultExtractorSetup virtuosoDefaultSetup
virtuosoDefaultTaper VLMDefaultSetup DFM LEFDefaultRouteSpec_gpdk045
minPRBoundaryInteriorHaloCG minProtrusionNumCutCG virtuosoDefaultSetup"
```

ciGetCPSelectedResults

```
ciGetCPSelectedResults(
     u_cache
     )
     => l_ciFinderResults / nil
```

Description

Returns a list of finder results selected in Circuit Prospector. Each item in the list is a finder result which contains the list of devices and the finder name.

Arguments

u_cache	The constraint cache ID.

Value Returned

l_ciFinderResults	List the finders found.
nil	No results found.

Example

```
ciGetCPSelectedResults(ciGetCellView())
((("/NM12" "inst")
     ("/NM0" "inst") properties
     (nil ciFinderName "Symmetric Instance Pairs - By Connectivity")
     )
)
```

ciGetDeviceBulkTermName

```
ciGetDeviceBulkTermName(
     d_deviceID
     )
     => t_terminalName / nil
```

Description

Returns the name of the bulk terminal for the passed device.

Arguments

d_deviceID	The device ID to return the bulk terminal for.

Value Returned

t_terminalName	Bulk terminal name for passed device.
nil	Command failed.

Example

```
ciGetDeviceBulkTermName(myMOSdevice)
"B"

ciGetDeviceBulkTermName(myBJTdevice)
"SUB"
```

ciGetDeviceInfo

```
ciGetDeviceInfo(
     d_deviceID
     t_deviceTypeName
     )
     => list / nil
```

Description

Returns the disembodied property list (DPL) associated with a particular device type.

Arguments

d_deviceID	The device ID for which the disembodied property list needs to be returned.
t_deviceTypeName	The device type associated with the given device.
	If the device type has not been registered then `nil` will be returned.

Value Returned

list	A DPL associated with the given device.
	Note: A disembodied property list is optional and will have been specified when the device type was registered with `ciRegisterDevice`. If a disembodied property list was not specified when the device type was registered then an empty list will be returned.
`nil`	Command failed.

Example

```
ciRegisterDevice("probeShort" list( list("analogLib"
         "res"
         "symbol"
         list(nil 'shortedTerminalMap list( list( "I0" "I1"))
               'shortTermExpr "ciGetParamValue(device \"r\") < 0.0001"
              (nil "pplusres" nil)))
devInfo = ciGetDeviceInfo(myRes "probeShort")
devInfo->shortedTerminalMap => ( ( "I0" "I1"))
devInfo->shortTermExpr => "ciGetParamValue(device \"r\") < 0.0001"
```

ciGetDeviceNames

```
ciGetDeviceNames(
     t_deviceName
     )
     => t_deviceNames / nil
```

Description

Returns all devices registered to the user-defined device name.

See also ciRegisterDevice.

Arguments

t_deviceName	Name of the device to return device names for.

Value Returned

t_deviceNames	The device names associated with the given device.
nil	No device names found.

Example

If `fet` is registered as:

```
ciRegisterDevice("fet"
    '( (nil "nmos" nil)
    '(nil "pmos" nil)
    '(nil "nmos3" nil)
    '(nil "pmos3" nil)
    '(nil "nmos4" nil)
    '(nil "pmos4" nil)))
```

then `ciGetDeviceNames("fet")` returns devices registered for "`fet`" as:

```
((nil "nmos" nil)
        '(nil "pmos" nil)
        '(nil "nmos3" nil)
        '(nil "pmos3" nil)
        '(nil "nmos4" nil)
```

```
        '(nil "pmos4" nil)
)
```

Also (as an example) for `nfet` and `pfet`:

```
ciRegisterDevice("nfet"
    '((nil "nmos" nil)
    '(nil "nmos1v" nil)
    '(nil "nmos1v_hvt" nil)
    '(nil "nmos1v_iso" nil)
    '(nil "nmos1v_nat" nil)
    '(nil "nmos2v" nil)
    '(nil "nmos2v_nat" nil)
    '(nil "nmos3" nil)
    '(nil "nmos4" nil)))

ciRegisterDevice("pfet"
    '((nil "pmos" nil)
    '(nil "pmos1v" nil)
    '(nil "pmos1v_hvt" nil)
    '(nil "pmos2v" nil)
    '(nil "pmos3" nil)
    '(nil "pmos4" nil)))
```

ciGetDeviceTermName

```
ciGetDeviceTermName(
     d_deviceId
     t_termName
     )
     => t_deviceTermName / nil
```

Description

Returns the terminal name of a device based on the device database ID and the user-defined parameter `termName`.

Arguments

d_deviceId	The device database ID.
t_termName	User-defined parameter mapped to the terminal names of the PDK using ciMapTerm.

Value Returned

t_deviceTermName	The name of the device terminal name.
`nil`	Command failed.

Example

Note: Matching expressions will not change automatically.

For example (see also ciMapTerm):

Note: The following uses a scenario where the fet in a user's PDK has **g=gate**, **s=source**, and **d=drain**. The objective of providing these examples is to clarify that `gate`, `myGate` , and `myFETGate` are all user-defined names and `g` is the terminal name.

- If you map `ciMapTerm("gate" '("g"))`

  Then you should use `ciGetDeviceTermName(device "gate")` in the matching expression.

  `[ciGetDeviceTermName(device "gate")` will return `"g"`

- If you map `ciMapTerm("myGate" '("g"))`

  Then you should use `ciGetDeviceTermName(device "myGate")` in the matching expression.

  `[ciGetDeviceTermName(device "myGate")`will return `"g"]`

- If you map `ciMapTerm("myFETGate" '("g"))`

  Then you should use `ciGetDeviceTermName(device "myFETGate")` in the matching expression

  `[ciGetDeviceTermName(device "myFETGate")`will return `"g" ]`

For **pre-defined structures** such as MOS Differential Pairs and MOS Inverters, `ciGetDeviceTermName` uses `gate`, `source`, and `drain` as the defaults for gate, source, and drain terminal names respectively. This will work successfully on Cadence PDK as the Cadence PDK fet device also has `gate`, `source`, and `drain` as the default terminal names.

Internally then, all pre-defined structures, like differential pairs and MOS invertors, perform a:

`ciGetDeviceTermName(device "gate")` which returns "`G`" by default

`ciGetDeviceTermName(device "source")` which returns "`S`" by default

`ciGetDeviceTermName(device "drain")` which returns "`D`" by default

For these structures to work on a user PDK, map:

`ciMapTerm("gate" '("`**g**`") )` then `ciGetDeviceTermName(device "gate")` will return "**g**".

With "`gate`" as the gate terminal name, the structures differential pair and MOS inverter will work correctly because "`gate`" is the gate fet terminal in the user PDK. Also, as the structures do not have anything in the matching expression (this defaults to `t`), there is nothing extra for to be performed here.

ciGetFinder

```
ciGetFinder(
     t_finderName
     )
     => l_finderAtributes / nil
```

Description

Returns a disembodied property list (DPL) of attributes associated with the specified finder. This DPL is the same that was used to register the finder using ciRegisterFinder.

Arguments

t_finderName	The name of the finder for which attributes need to be retrieved.

Value Returned

l_finderAtributes	The DPL of the specified finder and its attributes, which is of the following format:
	`list(nil 'name <name>`
	`'description <description>`
	`'iterator <iterator_name>`
	`'expression <expression>`
	`'defaultCGen <default_generator_name>`
	`'legalCGen <list_of_legal_generators>)`

`nil`	Command failed.

Example

```
ciGetFinder("Negative Supply")
```

The command for example might return the following:

```
(nil name "Negative Supply" description "Creates a group for each negative supply net"
 iterator "Net Iterator" expression "ciIsNet(net \"ground\")" defaultCGen
 "Negative Supply Route Priority" legalCGens nil
)
```

ciGetFirstDeviceTermName

```
ciGetFirstDeviceTermName(
     d_deviceID
     l_termNames
     )
     => s_termName / nil
```

Description

Returns the first terminal name on the specified device that matches one of the terminal names specified in the `termNames` list.

Arguments

d_deviceID	The device ID.
l_termName	A list of terminal names.

Value Returned

s_termName	Terminal name.
`nil`	Command failed.

Example

```
ciGetFirstDeviceTermName(myMOS list("gate" "base"))
"gate"
ciGetFirstDeviceTermName(myBJT list("gate" "base"))
"base"
```

ciGetFluidGuardRingDeviceEnum

```
ciGetFluidGuardRingDeviceEnum(
     )
     => t_fluidGuardRingDeviceString
```

Description

Returns a space delimited string of the fluid guard ring devices in the technology file. This string can be used to define an enum for constraint generators that require fluid guard ring device selection.

Also, see ciSetStructArgs and ciRegisterConstraintGenerator.

Arguments

None

Value Returned

t_fluidGuardRingDeviceString

A space delimited string of the fluid guard ring devices in the technology file.

Example

Get the fluid guard ring device names from the current technology file:

```
ciGetFluidGuardRingDeviceEnum() => "nring pring"
```

Then, register a constraint generator that has a fluid guard ring argument initialized to the names of the fluid guard ring devices in the current technology file:

```
ciSetStructArgs('MyGenerator
                 list(
                   list("Num Rows"                'int   1)
                   list("Add Guard Ring"          'bool  nil)
                   list("Fluid Guard Ring Device" 'enum
"ciGetFluidGuardRingDeviceEnum()")
                 )
               )
```

ciGetGenerator

```
ciGetGenerator(
     t_generatorName
     )
     => l_generatorAtributes / nil
```

Description

Returns a disembodied property list (DPL) of attributes associated to the specified constraint generator. This DPL is the same that was used to register the constraint generator using ciRegisterConstraintGenerator.

Arguments

t_generatorName	The name of the constraint generator for which attributes need to be retrieved.

Value Returned

l_generatorAtributes — The DPL of the specified constraint generator and its attributes, which is of the following format:

```
list(nil 'name <name>
    'description "description"
    'expression "expression"
    ['addToToolbar t]
    ['iconName "icon name"]
    ['args "argExprStr" | list( list(
    "argName" 'argType argVal1 argVal2 ...)]
    ['menu "menuName" | list("menuName1"
    "menuName2" "menuName3")]
    ['forcePopup t]
    ['preconditon "preconditionExpr"]
    ['callback "callbackExpr"]
    ['useCGenForEdit t]
    ['title "titleName"]
    ['size list(width, height)]
    ['templateName "templateName"]
    ['settings list(nil settingName value
...)
```

`nil`	Command failed.

Example

```
ciGetGenerator("Test Generator")
```

The command for example might return the following:

```
(nil name "Test Generator" description "Ask for Matching Variants"
    expression "ciRunMatchingConstraintsGenerator(args insts cache)"
    addToToolbar t iconName
    "templateMatched" args (
    ("enumparam" enum "low" "medium" "high")
    ("intparam" int 1 hide)
    ("floatparam" float 0.2)
    ("boolparam" bool t)
    ("stringparam" string "asd" callback "callbackExpr")
    )
    menu "Rapid Analog Prototype"
    forcePopup t
    precondition "!ciWithinConstraint('modgen cache
    ciGetMembersOfType(instsNetsPins 'inst))"
    callback "CbkExpr" useCGenForEdit t title "title"
    size (100 100) templateName "MOS_Cascode" settings
    (nil widgetPropertiesEnabled t))
```

ciGetIterator

```
ciGetIterator(
     t_iteratorName
     )
     => l_iteratorAtributes / nil
```

Description

Returns a disembodied property list (DPL) of attributes associated to the specified iterator. This DPL is the same that was used to register the iterator using ciRegisterIterator.

Arguments

t_iteratorName	The name of the iterator for which attributes needs to be retrieved.

Value Returned

l_iteratorAtributes	The DPL of the specified iterator and its attributes, which is of the following format:
	`list(nil 'name <name>`
	`'description <description>`
	`'iteratorFnName <iterator_function_name>`
	`'supportsFlattenedHier <t or nil>)`

`nil`	Command failed.

Example

```
ciGetIterator("Same Cell Iterator")
```

The command for example might return the following:

```
(nil name "Same Cell Iterator" description "Iterate over all devices with same cell
name. \nVariable \"device\" can be used in matchExpr" iteratorFnName
"ciSameCellIterator" supportsFlattenedHier t)
```

ciGetLAMComponentTypes

```
ciGetLAMComponentTypes(
     d_cv
     t_pdkLibName
     [ ?suppressRead g_suppressRead ]
     )
     => l_componentTypes
```

Description

Returns a list of the device, terminal, and parameter mappings specified in a `cph.lam` file associated with a particular PDK. This SKILL function can be used in conjunction with the ciCPRegistrationFromLAM SKILL function to create the device, terminal, and parameter registrations needed for the Circuit Prospector finders, iterators, and generators to function with the selected PDK.

Arguments

d_cv	The cellview that utilizes the PDK.
t_pdkLibName	The library name of the PDK.
?suppressRead *g_suppressRead*	A Boolean value that defaults to `nil` which controls whether the `cph.lam` files associated with the specified cellview should be read if they have not been read already.

Value Returned

l_componentTypes	A list containing the device, terminal, and parameter name mappings specified in the `cph.lam` files. **Note:** Even if there are problems, a list is returned, but the devices, terminals, and parameters properties in the list will be `nil`. For example: `list(nil libName <libName> devices nil terminals nil parameters nil)`

Example

```
gpdk045_compTypes = ciGetLAMComponentTypes(geGetEditCellView() "gpdk045")
```

```
(nil libName "gpdk045" devices (
    ("nfet"
    ("nmos1v" "nmos1v_3")
    )
    ("pfet"
    ("pmos1v" "pmos1v_3")
    )
)
terminals nil parameters (
    ("fingerWidth"
    ("fw")
    )
    ("lxActiveLayer"
    ("Oxide drawing")
    )
    ("lxMaxWidth"
    ("1e-06")
    )
)
)
info(ciCPRegistrationFromLAM(gpdk045_compTypes ?addNewLines t))
 ciRegisterDevice("nfet" append(ciGetDeviceNames("nfet") '(
 ("gpdk045" "nmos1v" nil)
 ("gpdk045" "nmos1v_3" nil)
)))
 ciRegisterDevice("pfet" append(ciGetDeviceNames("pfet") '(
 ("gpdk045" "pmos1v" nil)
 ("gpdk045" "pmos1v_3" nil)
)))
 ciRegisterDevice("fet" append(ciGetDeviceNames("fet") '(
 ("gpdk045" "nmos1v" nil)
 ("gpdk045" "nmos1v_3" nil)
)))
 ciRegisterDevice("fet" append(ciGetDeviceNames("fet") '(
 ("gpdk045" "pmos1v" nil)
 ("gpdk045" "pmos1v_3" nil)
)))
 ciMapParam("fingerWidth" append(ciGetParamMapping("fingerWidth") '("fw")))
 ciMapParam("lxActiveLayer" append(ciGetParamMapping("lxActiveLayer") '("Oxide
drawing")))
 ciMapParam("lxMaxWidth" append(ciGetParamMapping("lxMaxWidth") '("1e-06")))
```

ciGetMappedDeviceNames

```
ciGetMappedDeviceNames(
     )
     => l_regdDevices / nil
```

Description

Returns all default device names (fet, nfet, pfet, and BJT) that have been registered using `ciRegisterDevice`.

Arguments

None

Value Returned

l_regdDevices	A list of registered devices.
nil	Failed to find registered devices.

Example

```
ciGetMappedDeviceNames()
    ((mapName
    ((nil "pfinFET" nil)) name "pfin"
    )
    (mapName
    ((nil "nfinFET" nil)) name "nfin"
    )
)
```

ciGetNetNames

```
ciGetNetNames(
     t_netType
     )
     => (l_netNames | l_regexs | l_predicates) / nil
```

Description

Retrieves the net names, regular expressions and predicates that make up a net type.

See also:

- ciNetNames
- ciNetRegexs
- ciNetPredicates
- ciRegisterNetSuperType

Arguments

t_netType	A net type.

Value Returned

l_registeredNets	If `t_netType` is a known net type or super-type, returns a list containing separate lists of net names, regular expressions and predicates registered for `t_netType`. If there is only a single predicate, it is returned as an individual item. Multiple predicates are returned in a list.
`nil`	Command failed.

Example

```
;; Register net names, regular expressions and predicate.
ciRegisterNetNames("Power" '("vcc"))
ciRegisterNetRegexs("Power" '("[vV][dD][dD]"))
ciRegisterNetPredicate("Power" 'ciSigTypeMatchesNetType)
```

```
;; Retrieve them all in one go.
ciGetNetNames("Power") ; (("vcc") ("[vV][dD][dD]") ciSigTypeMatchesNetType)
```

ciGetNetSubTypes

```
ciGetNetSubTypes(
     t_superType
     )
     => l_subTypes / nil
```

Description

Retrieves the sub-types contained in a net super-type.

Arguments

t_superType	A super-type name.

Value Returned

l_subTypes	If *t_superType* is a super-type name, it returns the list of sub-type names registered for this super-type.
nil	Returns nil if *t_superType* is not a super-type name.

Example

```
ciRegisterNetSuperType("Supply" '("Power" "Ground"))
ciGetNetSubTypes("Supply") ; returns ("Power" "Ground")
```

Returns the sub-type names, `Power` and `Ground` registered for super-type, `Supply`.

ciGetNetSuperTypes

```
ciGetNetSuperTypes()
     => l_superTypes
```

Description

Returns the list of registered net super-types.

Arguments

None

Value Returned

l_superTypes	A list of registered net super-types.

Example

```
;; Register some super-types.
ciRegisterNetSuperType("Priority" '("High" "Low"))
ciRegisterNetSuperType("Supply" '("Power" "Ground"))

;; Retrieve all super-types.
ciGetNetSuperTypes() ; ("Priority" "Supply")
```

Returns the super-type names, `Priority` and `Supply`.

ciGetParamMapping

```
ciGetParamMapping(
     t_paramName
     )
     => l_paramNames / nil
```

Description

Returns all the parameters mapped for a given parameter name.

Arguments

t_paramName	A user-defined parameter name.

Value Returned

l_paramNames	The parameter names returned for a given parameter name.
`nil`	Command failed.

Example

If, for example, "`fetlength`" is mapped as:

```
ciMapParam("fetlength" '("l","len","length"))
```

Entering `ciGetParamMapping("fetlength")` will return:

```
("l" "len" "length")
```

ciGetParamName

```
ciGetParamName(
     d_deviceId
     t_paramPlaceholderName
     )
     => paramName / nil
```

Description

Returns the parameter name of the user-defined parameter that is mapped to the given parameter placeholder name.

Arguments

d_deviceId	The device database ID.
t_paramPlaceholderName	The placeholder name used to map the parameter name. This is the user-defined parameter that was used in the PDK which was mapped to the placeholder name by ciMapParam.

Value Returned

paramName	The parameter name.
nil	Command failed. Either no parameter was mapped or the mapped parameter is not set for the device.

Example

If a mapping for `fetLength` exists:

```
ciMapParam(“fetLength” ‘(“l”))
```

The device is also set as:

```
device=car(geGetSelSet())
```

Then:

```
ciGetParamName(device “fetLength”)
```

will return “`l`” as the name of the parameter mapped to the placeholder name of “`l`”.

ciGetParamValue

```
ciGetParamValue(
     d_deviceId
     t_parameterPlaceholderName
     [ ?path t_path ]
     [ ?convertNumberStrings g_convertNumberStrings ]
     )
     => paramValue / nil
```

Description

Returns the parameter value for the user-defined parameter that is mapped to the given parameter placeholder name.

Arguments

d_deviceId	The database ID of the device.
t_parameterPlaceholderName	The placeholder name that is used to map the parameter name The user-defined parameter, that is used in the PDK, was mapped to the placeholder name by ciMapParam.
?path *t_path*	Optional instance path to allow `pPar` expressions to be evaluated.
?convertNumberStrings *g_convertNumberStrings*	Optional flag to control whether number strings should be converted to numbers to allow number comparison rather than string comparison. For example, this allows `"4e-07"` and `"400n"` to be compared correctly.

Value Returned

paramValue	The value of the mapped parameter for the device.
nil	Command failed.

Example

If a mapping for the placeholder “`fetlength`” has been set by:

```
ciMapParam("fetLength" '("l"))
```

and the device is defined using the following from the cellview:

```
device=car(geGetSelSet())
```

Then, the following will return the value of the parameter “`l`” mapped to the “`fetlength`” placeholder and set for the device:

```
ciGetParamValue(device "fetlength")
"340.0n" ==> returns the fetlength for that device
ciGetParamValue(device "fetLength" ?convertNumberStrings t)
3.4e-07 ==> returns the fetlength for that device
```

ciGetParamValueOrDefault

```
ciGetParamValueOrDefault(
     d_obj
     t_paramName
     g_defValue
     [ ?warnUnmapped g_warnUnmapped ]
     [ ?warnUnfound g_warnUnfound ]
     [ ?aelEnv g_aelEnv ]
     )
     => d_paramVal
```

Description

Retrieves the value of the named parameter on the passed database object. The parameter name should have been registered ciMapParam. If the parameter is not found then the default value will be returned.

Arguments

d_obj	The database ID of the object to retrieve the parameter from.
t_paramName	The name of the parameter.
d_defValue	The default value to be returned if the parameter is not found.
?warnUnmapped *g_warnUnmapped*	Displays a warning message if the parameter is not mapped using ciMapParam. Default: `t`
?warnUnfound *g_warnUnfound*	Displays a warning message if the parameter is not found on the database object. Default: `t`
?aelEnv *g_aelEnv*	If specified, use the `aelEnv` to resolve `iPar` and `pPar` expressions. See also, aelEnvCreate() and aelSetLineage(). Default: `nil`

Value Returned

d_paramVal	Returns the parameter value or defValue if the parameter is not found.

Examples

```
ciGetParamValueOrDefault(inst "mFactor" 1)
=> 2 ;;; parameter found
ciGetParamValueOrDefault(inst "XXX" 33.3)
=> 33.3 ;;; parameter not found.
```

ciGetParamValues

```
ciGetParamValues(
     d_deviceId
     l_parameterNames
     [ ?path t_path ]
     [ ?asDPL g_asDPL ]
     )
     => l_paramValues / nil
```

Description

Returns the values of the specified parameters for the specified device. By default, the values are returned as a disembodied property list (DPL), but can also be returned as a simple list.

See also ciMapParam for specifying PDK independent device parameter name mappings.

Arguments

d_deviceId	The database ID of the device.
t_parameterNames	A list of the parameter names for which values need to be retrieved. The parameter names should be those that have been registered using the ciMapParam SKILL command.
?path *t_path*	Specifies the hierarchical instance path to the device to allow the `pPar` expressions to be evaluated. This is an optional argument.
?asDPL *g_asDPL*	Determines whether the parameter values should be returned as a DPL or simple list. This is an optional argument. Default: `t`

Value Returned

l_paramValues	Returns either a DPL or a simple list containing the requested parameter values.
`nil`	Shows that the command failed.

Examples

- Returns a DPL of values of the `mFactor`, `length`, and `width` parameters for `dev2`:

```
IresDPL = ciGetParamValues(dev2 '("mFactor" "length" "width"))
                (nil mFactor "1" length "180.0n" width "2u")
          resDPL->mFactor
                 "1"
```

- Returns a simple list of values of the `mFactor`, `length`, and `width` parameters for `dev3` that is on the instance path `/I1/I2`:

```
resL = ciGetParamValues(dev3 '("mFactor" "length" "width") ?asDPL nil ?path “/
I1/I2”)
("1" "180.0n" "3u")
car(resL)
"1"
```

ciGetStructure

```
ciGetStructure(
     t_structureName
     )
     => l_structureAtributes / nil
```

Description

Returns a disembodied property list (DPL) of attributes associated to the specified structure. This DPL is the same that was used to register the structure using ciRegisterStructure.

Arguments

t_structureName	The name of the structure for which attributes needs to be retrieved.

Value Returned

l_structureAtributes

The DPL of the specified structure and its attributes, which is of the following format:

```
list(nil 'name <name>
    'description <description>
    'matchExpr "match Expression"
    'finder "finder name"
    ['nets list(list(nil 'name "netName"
      'expr "netExpr") ...)]
    ['insts list(list(nil 'instName|'instId
      "instVal" 'expr "exprVal" 'terms
      list(list(nil 'name ['expr] ['net]
      ['repeatType] ['repeatConn]) ...)
    ...))
    ['pins list(list(nil 'name "pinName" 'net
      "pinNet") ...)]
    ['constraints list(list(nil 'name
    "name"'type "typeVal" 'params
    list(list(nil 'name 'val 'type) ...)
    'members list(list(nil 'name 'type 'index
    'params) ...)) ...)
```

`nil` Command failed.

Example

```
ciGetStructure("Test Structure")
```

The command for example might return the following:

```
(nil name "Test Structure" type "t1"
 description "description" matchExpr "matchExpr" finder
 "Negative Supply" nets (
     (nil name "net1" expr "netExpr")
 ) insts (
     (nil instId 1 expr "instExpr")
     (nil instName "123" expr "expr1"
     terms
     ((nil name "t1" expr "e1"
    net "n1" repeatType "r1" repeatConn
    "rn"
    )
     ) repeatable "repInsts"
     )
 )
 pins (
     (nil name "pinName" net "pinNet")
) constraints (
     (nil name "con1" type "symmetry"
     params
     ((nil name "p1" val 1
    type "numType"
    )
     ) members
     ((nil name "n1" type "n1"
    index 1 params
    ((nil name "mp2" val "mpVal"
        type "string"
        )
    )
    )
   )
   )
 )
 )
```

ciGetTechFile

```
ciGetTechFile()
     => d_techFile
```

Description

Retrieves the technology file for the current window.

Arguments

None

Value Returned

d_techFile	The database ID of the technology file or nil.

Example

```
tf = ciGetTechFile()
tf~>libName
"gpdk045"
```

ciGetTechMPPNames

```
ciGetTechMPPNames()
     => l_mppNames / nil
```

Description

Returns a list of the MPP names for the current technology file. The MPP names are used in the creation of guard rings.

Arguments

None

Value Returned

l_mppNames	List of the MPP names in the technology file.
`nil`	There were no MPP names found in the technology file.

Example

```
ciGetTechMPPNames()
=> ("N-Tap" "P-Tap")
```

ciGetTermNames

```
ciGetTermNames(
     t_termName
     )
     => terminalNames
```

Description

Returns all the terminal names mapped to the user-defined parameter, `termName`.

Arguments

t_termName	The terminal name parameter.

Value Returned

terminalNames	The terminal names associated with the given `termName`. **Note:** If no mapping is found the returned terminal name will be the same as the defined parameter `termName`.

Example

If the terminal name mapping is set as:

```
ciMapTerm(“myGate” “G”)
```

then

```
ciGetTermNames(“myGate”)
```

will return `"G"` as the mapped terminal name.

ciGuardRingForCluster

```
ciGuardRingForCluster(
     u_cache
     l_instances
     [ ?mppName t_mppName ]
     )
     => l_guardRing / nil
```

Description

Used by the *Circuit Prospector* assistant as a generator to set a *Guard Ring* constraint to an existing *Cluster* constraint.

The *Guard Ring* constraint is created only if the name of the net can be found.

The name of the net for the guard ring is determined by the net connected to the `bulk` terminal of the first instance of the given list of instances.

Note: An explicit `bulk` terminal must be registered with the `ciMapTerm` function, while an implicit `bulk` terminal can be registered with the `ciMapParam` function.

Arguments

u_cache	Constraint cache that contains the *Cluster* constraints to be iterated, and in which the *Guard Ring* constraints must be created.
l_instances	List of members of the same *Cluster* constraint that are selected in the *Circuit Prospector* assistant's browser.
?mppName *t_mppName*	Optional argument. Name of a multipart path (MPP) template to be used by the guard ring. By default, the name of the MPP is the first one defined in the technology file.

Value Returned

l_guardRing	A list made of the constraint ID for the new *Guard Ring* constraint when it is created.
nil	`No actions completed.`

Example

If `NM1` and `NM2` are the names of two instances that are members of an existing *Cluster* constraint, the following functions creates a *Guard Ring* and the member of that new *Guard Ring* is the *Cluster* constraint for `NM1` and `NM2`:

```
cache = ciCacheGet(geGetEditCellView()
ciGuardRingForCluster(cache list( list("/NM1" inst) list("/NM2" inst)))
```

ciGuardRingForModgen

```
ciGuardRingForModgen(
     u_cache
     l_instances
     [ ?mppName t_mppName ]
     )
     => l_guardRing / nil
```

Description

Used by the *Circuit Prospector* assistant as a generator to set a *Guard Ring* constraint to an existing *Modgen* constraint.

The *Guard Ring* constraint is created only if the name of the net can be found.

The name of the net for the guard ring is determined by the net connected to the `bulk` terminal of the first instance of the given list of instances.

Note: An explicit `bulk` terminal must be registered with the `ciMapTerm` function, while an implicit `bulk` terminal can be registered with the `ciMapParam` function.

Arguments

u_cache	Constraint `cache` that contains the *Modgen* constraints to be iterated, and in which the *Guard Ring* constraints must be created.
l_instances	List of members of the same *Modgen* constraint that are selected in the *Circuit Prospector* assistant's browser.
?mppName *t_mppName*	Optional argument. Name of a multipart path (MPP) template to be used by the guard ring. By default, the name of the MPP is the first one defined in the technology file.

Value Returned

l_guardRing	A list made of the constraint ID for the new *Guard Ring* constraint when it is created.
nil	`No actions completed.`

Example

If `NM1` and `NM2` are the names of two instances that are members of an existing *Modgen* constraint, the following functions creates a *Guard Ring* and the member of that new *Guard Ring* is the *Modgen* constraint for `NM1` and `NM2`:

```
cache = ciCacheGet(geGetEditCellView()
ciGuardRingForModgen(cache list( list("/NM1" inst) list("/NM2" inst)))
```

ciHaveSameParamValues

```
ciHaveSameParamValues(
     l_devices
     l_paramNames
     [ ?path t_path ]
     )
     => t / nil
```

Description

Returns `t` if the specified parameters have the same values on all the passed devices. The parameter names should be the PDK independent parameter names specified using the ciMapParam SKILL command.

Arguments

l_devices	List of database IDs of the required devices.
t_parameterNames	A list of the required parameter names that are the same as registered using the ciMapParam SKILL command.
?path *t_path*	Specifies the hierarchical instance path required for the evaluation of the `pPar` expressions. This is an optional argument.

Value Returned

t	All the required parameter values match on all of the listed devices.
nil	Some parameter values do not match on some of the listed devices.

Examples

Assume the `ciGetParamValues` SKILL command returns the following values of the `mFactor`, `length`, and `width` parameters for `dev1`, `dev2`, and `dev3`:

```
ciGetParamValues(dev1 '("mFactor" "length" "width"))=>  (nil mFactor "1" length
"180.0n" width "2u")
ciGetParamValues(dev2 '("mFactor" "length" "width"))=>  (nil mFactor "1" length
"180.0n" width "2u")
```

```
ciGetParamValues(dev3 '("mFactor" "length" "width"))=>  (nil mFactor "4" length
"180.0n" width "2u")
```

Based on the values returned above, the results of the `ciHaveSameParamValue` SKILL command vary as following for `dev1`, `dev2`, and `dev3`:

```
ciHaveSameParamValues(list(dev1 dev2) '("mFactor" "length" "width")) => t
```

```
ciHaveSameParamValues(list(dev1 dev2 dev3) '("mFactor" "length" "width") ?path “/
I1/I2”) => nil
```

ciHierarchicalSeriesIterator

```
ciHierarchicalSeriesIterator(
     d_cellviewID
     t_matchExpression
     )
     => l_devices
```

Description

(ICADVM20.1 EXL Only) Returns a list of corresponding devices that match the series structures in the cellview. If the finder match net expression evaluates to `nil`, the structures are ignored.

Arguments

d_cellviewID	The cellview to apply the iterator to.
t_matchExpression	The expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_devices	List of devices that match the structure in schematic.

Example

```
finder = ciGetFinder("APR Active Same Cell Stacked Device Iterator")
insts = ciHierarchicalSeriesIterator(geGetEditCellView() finder->expression) ("/
I1" "/I2")
```

ciIgnoreDevice

```
ciIgnoreDevice(
     d_device
     )
     => t / nil
```

Description

This is a utility function that can be used in finder or constraint generator expressions. It returns `t` if the passed device should be ignored.

For example, `device->master->nlAction == "ignore"` or `device->lvsIgnore == "TRUE"`.

Arguments

d_device	The device to be checked. This is the `dbId` of the database object.

Value Returned

`t`	Device is ignored.
`nil`	Device is not ignored.

Examples

```
dev1->master->nlAction
=> nil
ciIgnoreDevice(dev1)
=>nil
dev1->master->nlAction = "ignore"
ciIgnoreDevice(dev1)
=> t
```

ciInstGetSplitFingers

```
ciInstGetSplitFingers(
     u_cache
     t_instancePath
     )
     => g_split
```

Description

Returns a Boolean value to identify whether the specified instance has been set to use Split Fingers. It can be used by a constraint generator to control how the layout is generated for its member instances.

Arguments

u_cache	The constraint cache.
t_instancePath	The device name.

Value Returned

g_split	Returns `t` if the instance has been set to use Split Fingers or `nil` if not.

Example

Check if an instance will use Split Fingers:

```
ciInstSetSplitFingers(ciGetCellView() "M4" t)
ciInstGetSplitFingers(ciGetCellView() "M4")
  => t
```

Confirm Split Fingers has not been set for an instance:

```
ciInstSetSplitFingers(ciGetCellView() "M4" nil)
ciInstGetSplitFingers(ciGetCellView() "M4")
  => nil
```

ciInstIterator

```
ciInstIterator(
     d_cellview
     t_matchExpr
     [ l_hierCellViews ]
     )
     => list / nil
```

Description

Used by *Circuit Prospector* finders to iterate over all design instances, collating them into groups based on the result of evaluating the passed (match) expression.

The `ciInstIterator` function evaluates the match expression with the current design instance that is assigned to a local variable named `device`. The device instance can then be referenced in the `matchExpr` (as shown in the example below).

Note: If the match expression evaluates to `nil` the device will be ignored.

Arguments

d_cellview	The cellview containing the design instances to be iterated over.
t_matchExpr	The matched expression string to be used in the iteration.
l_hierCellViews	Optional argument that can contain a list of cellview IDs and their occurrence paths. If set, this then allows the iterator to iterate over all instances in the hierarchy in one run. This is required for when a finder is run on a flattended hierarchy (see *Run finder*...). The default setting is `nil`.

Value Returned

list	List of returned instances.
	For example:
	`list(`
	`    list( inst1 inst2 inst3 ...)`
	`    )`

`nil`	Device ignored.

Example

To group all FETs:

```
finderDevExpr = "ciIsDevice(device "fet")"
fetGroups = ciInstIterator(cv finderDevExpr)
print(fetGroups~>name)
("fet1" "fet2" "fet3")
```

Example using `hierCellViews` argument:

```
fetGroups = ciInstIterator(cv finderDevExpr list( list( nil `hierPath "" `cellview
cellview) ))
```

ciInstListSplitFingers

```
ciInstListSplitFingers(
     u_cache
     )
     => l_instancePaths
```

Description

Returns the names of the instances that use Split Fingers. It can be used by a constraint generator to control how the layout is generated for its member instances.

Arguments

u_cache	The constraint cache.

Value Returned

l_instancePaths	Lists the instances that use Split Fingers.

Example

Get a list of instances that use Split Fingers:

```
ciInstListSplitFingers(ciGetCellView())
  => ("M4" "I2/M4")
```

Stop using Split Fingers for all currently registered instances:

```
cache = ciGetCellView()
  mapcar( lambda( (instname) ciInstSetSplitFingers(cache instname nil) )
         ciInstListSplitFingers(cache))
```

ciInstSetSplitFingers

```
ciInstSetSplitFingers(
     u_cache
     t_instancePath
     g_split
     )
     => t / nil
```

Description

Specifies whether an instance uses Split Fingers, that is, a separate instance per finger in layout. It can be used by a constraint generator to control how the layout is generated for its member instances.

Arguments

u_cache	The constraint cache.
t_instancePath	The device name.
g_split	Specify `t` if the instance should use Split Fingers and `nil` if it should not.

Value Returned

`t`	The instance uses Split Fingers.
`nil`	The instance does not use Split Fingers.

Example

Use Split Fingers for instance `M4` in the current cellview or a lower-level cellview `I2/M4`:

```
ciInstSetSplitFingers(ciGetCellView() "M4" t)
ciInstSetSplitFingers(ciGetCellView() "I2/M4" t)
```

Stop using Split Fingers for an instance:

```
ciInstSetSplitFingers(ciGetCellView() "M4" nil)
ciInstSetSplitFingers(ciGetCellView() "I2/M4" nil)
```

ciInstsNetsPinsFromSelSet

```
ciInstsNetsPinsFromSelSet(
    )
    => l_selectedObjList / nil
```

Description

Returns a list of instances, nets, pins, or instTerms (`InstsNetsPins`) for the current selection.

Arguments

None

Value Returned

l_selectedObjList	Lists of instances, nets, pins, or instTerms for the current selection.
nil	No instances, nets, pins, or instTerms were found for the current selection.

Example

With an instance, net, pin, and instTerm selected, call

```
ciInstsNetsPinsFromSelSet()
```

Returns a list similar to the following:

```
(("C4:G" instTerm)
    ("C4" inst)
    ("SIDDQ:P__13" pin)
    ("bias1" net)
)
```

ciInstTermIterator

```
ciInstTermIterator(
     d_cellview
     t_finderInstTermExpr
     )
     => l_instList / nil
```

Description

Used by the *Circuit Prospector* to iterate over all design instance terminals in the passed `cellview`. Instance terminals (`instTerms`) are grouped together based on the result of evaluating the passed expression (`finderInstTermExpr`).

- All `instTerms` that have the same `finderInstTermExpr` result are grouped together.
- If the `finderInstTermExpr` evaluates to `nil`, then the `instTerm` is ignored.

Arguments

d_cellview	The cellview containing the design instance terminals to be iterated over.
t_finderInstTermExpr	The `finderInstTermExpr` can reference `term` (the `instTerm`) and `inst` (the instance associated with that `instTerm`).

Value Returned

l_instList — List of returned instances.

For example:

```
list(
    list(d_instTerm0 .. d_instTermN)
    list(d_instTerm0 ..d_instTermM)
    )
```

`nil` — Instance terminal ignored.

Example

To return groups of `instTerms` connected to the same net:

```
ciInstTermIterator(cv "term->net")
```

ciIsDevice

```
ciIsDevice(
     d_deviceId
     t_deviceName
     )
     => t / nil
```

Description

Checks whether a device belongs to a particular device type.

The `ciIsDevice` command can also search for a device based on its `cellName`. This *smart* search is only performed when the device for a particular type is not registered.

For example, if `nfet` is not registered, then `ciIsDevice(device nfet)` will check if `device=>cellName` has a `nmos` in its `cellName`.

Note: If `nfet` is registered, then `device=>cellName` will match the devices registered with `nfet`.

Arguments

d_deviceId	The device to be checked. This is the `dbId` of the database object.
t_deviceName	The device type to be checked against. This is a device type that has been registered using ciRegisterDevice.

Value Returned

`t`	Device is specified type.
`nil`	Device is not specified type.

Example

If, for example, fet is registered as:

```
ciRegisterDevice("fet"
    '((nil "nmos" nil)
    '(nil "pmos" nil)
```

```
    '(nil "nmos3" nil)
    '(nil "pmos3" nil)
    '(nil "nmos4" nil)
    '(nil "pmos4" nil)))
```

And device is set, for example, as:

```
device=car(geGetSelSet())
```

Then:

```
ciIsDevice(device "fet")
```

returns `t` if device is of type "`fet`", else `nil`.

ciIsNet

```
ciIsNet(
     d_netId
     t_netType
     [ ?regexIgnoreBundles g_regexIgnoreBundles ]
     )
     => t / nil
```

Description

Checks whether a particular `netId` belongs to a particular `netType`.

Arguments

d_netId	The net to be checked. This is the `dbId` of the database object.
t_netType	The net type to be checked against. This is a net type that has been registered using ciRegisterNet.

`?regexIgnoreBundles` *g_regexIgnoreBundles*

Optional Boolean argument that defaults to `t`. This argument controls whether bundle nets should be split into their component parts to see if any component part matches the net type. The bundle names can match different net types. For example, for the bundle net `"vdd,gnd"`, calling `ciIsNet()` will return `t` for net types `"power"` and `"ground"`.

Value Returned

`t`	Net is specified type.
`nil`	Net is not of specified type.

Example

If, for example, "`supply`" type nets are registered as:

```
ciRegisterNet("supply" '("vcc" "vdd" "vss"))
```

And `net` is defined as:

```
myNet=car(geGetSelSet())~>net
```

Then, the following statement will return `t` if `myNet` is of type "`supply`", else `nil`:

```
ciIsNet(myNet "supply")
```

ciListAllCategoryNames

```
ciListAllCategoryNames(
     [ g_includeSeparator ]
     [ g_listDisabled ]
     )
     => l_categoryNames
```

Description

Lists all the categories in Circuit Prospector that have been registered by using ciRegisterAssistant. By default, it only lists the enabled assistants.

Arguments

g_includeSeparator	If set to `t`, result of this SKILL function also includes the separators between category names. By default, it is set to `nil`.
g_listDisabled	If set to `t`, result of this SKILL function also includes the names of disabled categories. By default, it is set to `nil` and returns only the enabled categories.

Value Returned

l_categoryNames	A list of category names.

Example

- Returns only the enabled categories:

```
ciListAllCategoryNames()
("Rapid Analog Prototype" "Structures" "Devices" "Nets" "Pins" "Inst Terms")
```

- Returns both enabled and disabled categories including separators:

```
ciListAllCategoryNames(t t)
("Rapid Analog Prototype" "Structures" "Devices" "Nets" "Pins"
"Inst Terms" nil "Properties" nil "Electrical Constraints"
"Placement Constraints" "Routing Constraints" nil "Clusters"
"Matched Parameters" "Modgens" "Orientations" "Symmetries")
```

ciListAllFinderNames

```
ciListAllFinderNames()
     => l_finderNames
```

Description

Lists all the finders in Circuit Prospector that have been registered by using ciRegisterFinder.

Arguments

None

Value Returned

l_finderNames	A list of finder names.

Example

Returns all the finder names:

```
ciListAllFinderNames()
("Active Same Cell" "Active Same Cell and Size" "Active Same Cell and Finger
Width" "Active Same Cell and Common Gate" "Active Same Cell and Common Bulk"
"Active Common Gate" "Active Same Cell, Common Gate and Bulk" "Passive Same
Cell" "Passive Same Cell and Size" "Supply Nets" "Non-Supply Nets" "Symmetric
Nets" "Bus and Bundle" "Registered (sigType)" "Same sigType (none)"
"Pins (Symmetry By Connectivity)" "Top Pins (Alignment)" "Bottom Pins
(Alignment)" "Left Pins (Alignment)" "Right Pins (Alignment)" "Supply Pins
(Alignment)" "Input Pins (Alignment)" "Output Pins (Alignment)" "InstTerms by
Instance" "InstTerms by Name" "InstTerms by Net" "Registered Nets (sigType)"
"Same sigType Nets (none)" "fets (variantInfo)" "Same variantInfo Devices"
"pfets (placerControlledWell)" "Same placerControlledWell Devices"
"Symmetric Pairs (Correlation)" "Same Cell,Size,Bulk Devices (Matched
Parameters)" "Common Bulk (Cluster)" "Digital Cells (Cluster)"
"Same Cell,Size,Bulk Devices (Modgen)" "Clusters (Cluster Boundary)"
"Clusters (Guard Ring)" "Modgens (Guard Ring)" "Same Device Type (Orientation)"
"Modgens (Orientation)" "Inverters (Alignment)" "Inverters (Distance)"
"Supply Nets (Net Priority)" "Bus Bits" "Instances (Symmetry By Pre-selection)"
"Instances (Symmetry By Connectivity)" "Nets (Symmetry By Connectivity)"
```

```
"Symmetric Pairs (Matched Parameters)" "Same Cell,Finger Width MOS (Matched
Parameters)" "Current Mirror (Matched Parameters)" "Same Cell and Size
(Cluster)" "Same Cell,Finger Width MOS (Cluster)" "Digital Cells (Modgen)"
"Orientation as in Schematic" "MOS Current Mirror" "MOS Cascoded Current
Mirror" "MOS Cascoded Current Mirror2" "MOS Cascode"
"MOS Transmission Gate" "MOS Differential Pair" "MOS Differential Pair - Cross
Coupled" "MOS Common Gate" "MOS Parallel" "MOS Active Load" "MOS Inverter"
"Symmetric Instance Pairs - By Pre-selection" "Symmetric Instance Pairs - By
Connectivity" "Self Symmetric Instances - By Connectivity" "Placement Path"
"Passive Arrays" "Active Same Cell large mfactor" "Capacitor Cluster" "Vertical
Orientation" "Negative Supply" "Positive Supply" "Enforce Precedence"
)
```

ciListAllGeneratorNames

```
ciListAllGeneratorNames()
     => l_generatorNames
```

Description

Lists all the constraint generators in Circuit Prospector that have been registered by using ciRegisterConstraintGenerator.

Arguments

None

Value Returned

l_generatorNames	A list of constraint generator names.

Example

Returns all the constraint generator names:

```
ciListAllGeneratorNames()
("Matched Parameters" "Matched Size" "Symmetry" "vSymmetry" "hSymmetry"
"Alignment" "Orientation" "Module" "Same Cell Name - Group + Relative Orientation"
"Common Gate - Group + Relative Orientation" "Same Well - Group + Relative
Orientation" "Module + Matched Parameters" "Group + Relative Orientation + Vertical
Alignment" "Group + Relative Orientation + Horizontal Alignment" "Group + Relative
Orientation + Matched Parameters" "Symmetry + Matched" "Matched + Distance"
"Matched + Symmetry + Distance" "Group" "Supply Route Priority" "Non Supply Route
Priority" "Matching (strength)" "Common Centroid" "Inst Symmetry" "Net Symmetry"
"Same Orientation" "Orientation per Device Type" "Bus for Nets" "Bus for Signal
Bits" "sigType for registered nets" "None" "Alignment for Right Pins" "Alignment
for Bottom Pins" "Alignment for Left Pins" "Alignment for Top Pins" "variantInfo
for FETs" "placerControlledWell" "Correlation" "Matched Parameters for Same Cell
Size and Bulk" "Cluster for Devices with common bulk" "Cluster for Standard Cells"
"Modgen for Same Cell Size and Bulk" "Cluster Boundary for Cluster" "Guard Ring for
cluster" "Guard Ring for modgen" "Orientation for modgen" "Alignment for Inverters"
"Distance for Inverters" "Symmetry for Pins" "Symmetry for Nets" "Symmetry for
Instance" "MatchedParameters for Symmetric Instances" "Matched Parameters for
Current Mirror" "Cluster for Devices with same name and size" "Cluster for Devices
```

```
with same finger width" "Modgen for Standard Cells" "Distance + Matched Orientation
+ Vertical Alignment" "Common Bulk - Group + Matching Orientation" "Matched
Parameters for Same Finger Width" "Placement Path" "Module (Cascode MOS
Transistors)" "Module (Cascoded Current Mirror)" "Module (Current Mirror)" "Module
(Diff Pair)" "Module (Passive Device Array)" "Module (Large mfactor)" "Symmetry
(default axis)" "Orientation Vertical" "Negative Supply Route Priority" "Positive
Supply Route Priority" "Enforce Modgen Symmetry Precedence" "GenericModgen" "MOS
Active Load - Matched + Symmetry + Distance")
```

ciListAllIteratorNames

```
ciListAllIteratorNames()
     => l_iteratorNames
```

Description

Lists all the iterators in Circuit Prospector that have been registered by using ciRegisterIterator.

Arguments

None

Value Returned

l_iteratorNames	A list of iterator names.

Example

Returns all the iterator names:

```
ciListAllIteratorNames()
("Same Cell Iterator" "Pin Iterator" "Net Iterator" "Instance Iterator" "InstTerm
Iterator" "XY Symmetric Iterator" "XY Net Symmetric Iterator" "MOS Current Mirror
Iterator" "MOS Cascoded Current Mirror Iterator" "MOS Cascoded Current Mirror
Iterator2" "MOS Cascode Iterator" "MOS Transmission Gate Iterator" "MOS
Differential Pair Iterator" "MOS Differential Pair - Cross Coupled Iterator" "MOS
Common Gate Iterator" "MOS Parallel Iterator" "MOS Active Load Iterator" "MOS
Inverter Iterator" "XY Instance Symmetric Iterator" "Signal Iterator" "Bundle
Signals Iterator" "XY Pin Symmetric Iterator" "Current Path Iterator")
```

ciListAllStructureNames

```
ciListAllStructureNames(
    )
    => l_structureNames
```

Description

Lists all the structures in Circuit Prospector that have been registered by using ciRegisterStructure.

Arguments

None

Value Returned

l_structureNames	A list of structure names.

Example

Returns all the structure names:

```
ciListAllStructureNames()
("Test Structure")
```

ciListGeneratableConstraintGroups

```
ciListGeneratableConstraintGroups(
    g_ciCon
    t_CGDefName
    )
    => l_groupNames
```

Description

The default generatable constraint group listing callback used by ciSetCMCGSKILLCallbacks when no user-defined function has been specified.

Value Returned

l_groupNames	A list of the generatable constraint group names.

Example

See ciSetCMCGSKILLCallbacks

ciMakeHierContext

```
ciMakeHierContext(
     l_currentHierInfo
     l_allHierInfo
     )
     => disembodied_property_list / nil
```

Description

Used by Circuit Prospector iterators that are required to run on flattened hierarchies. The iterators need to define a local variable, hierContext, to be referenced by finders that use that iterator. This function is used to initialize that local variable.

Arguments

l_currentHierInfo	A disembodied property list, with hierPath being the hierarchical path to the current cell, and cellview being the current cellview.
l_allHierInfo	A disembodied property list of all cellviews and paths in the design.

Value Returned

disembodied_property_list	List with members of currentHier and allHiers
nil	Context status not returned.

Example

```
currentHierCellViewInfo = list(nil 'hierPath geGetInstHier(getCurrentWindow()
'cellview geGetEditCellView())
hierContext = ciMakeHierContext(curentHierCellViewInfo hierCellViews)
```

ciMakeObjectInfo

```
ciMakeObjectInfo(
     d_object
     l_cvInfo
     )
     => l_listObj
```

Description

A utility function for constructing a disembodied property list to be used by *Circuit Prospector* iterators to return hierarchical path information about the found objects.

Arguments

d_object	Object ID.
l_cvInfo	List of cellview information.

Value Returned

l_listObj	Disembodied property list.

Example

```
ciMakeObjectInfo(object cvInfo)
list(nil 'dbId object 'hierPath "/I15/I21")
```

ciMapParam

```
ciMapParam(
     t_paramPlaceholderName
     l_paramMapNameList
     )
     => t / nil
```

Description

Maps a parameter placeholder name to the parameter names used in the PDK.

See also Setting Match Subset Parameter Values.

Note: Placeholder names starting with a lowercase letter are reserved for scanning functions defined by Cadence. In particular, the following placeholder names are used by existing iterators and finders:

- `width`
- `length`
- `fingerCount`
- `fingerWidth`
- `mFactor`
- `resValue`
- `capValue`
- `indValue`
- `area`
- `perimeter`
- `segments`

Arguments

t_paramPlaceholderName	The parameter placeholder name.
l_paramMapNameList	The parameter name mapping list.

Value Returned

`t`	Successful parameter mapping.
`nil`	Parameter mapping failed.

Examples

```
ciMapParam("width"      append(ciGetParamMapping("width")       '("w" "Width")))
ciMapParam("length"     append(ciGetParamMapping("length")      '("l" "Length")))
```

ciMapTerm

```
ciMapTerm(
     t_termName
     l_termMapNameList
     )
     => t / nil
```

Description

Maps the user-defined `termName` to the terminal names used in the PDK.

Note: The following reserved placeholders for terminal names have been created by Cadence for customization use:

- `gate`
- `source`
- `drain`
- `bulk`
- `base`
- `emitter`
- `collector`

Arguments

t_termName	A user-defined terminal name string. **Note:** Mapping is done to the terminal names used in the design.
l_termMapNameList	The terminal mapping name list.

Value Returned

`t`	Successful terminal mapping.
`nil`	Terminal mapping failed.

Example

```
ciMapTerm("drain" '("d" "D" "drain" "DRAIN" "Drain"))
ciMapTerm("gate" '("g" "G" "gate" "GATE" "Gate"))
ciMapTerm("source" '("s" "S" "source" "SOURCE" "Source"))
ciMapTerm("bulk" '("S" "BULK" "well" "SUB" "B"))
ciMapTerm("collector" '("c" "C" "collector"))
ciMapTerm("base" '("b" "B" "base" "BASE" "Base"))
ciMapTerm("emitter" '("e" "E" "emitter"))
ciMapTerm("resValue" '("r"))
ciMapTerm("indValue" '("I"))
ciMapTerm("capValue" '("c"))
```

Note: The matching expression will not change automatically. For more information see the examples in ciGetDeviceTermName.

ciMatchedFingerWidth

```
ciMatchedFingerWidth(
     u_cache
     l_instances
     )
     => nil
```

Description

Used by the *Circuit Prospector* assistant as a generator to set a *Matched Parameters* constraint to the selected instance members.

The parameter set for matching is the one registered as *fingerWidth* with the function `ciMapParam`.

The *Matched Parameters* constraint is created either when no such constraint exists for at least one member, or the new members are added to the *Matched Parameters* constraint when it exists for the same parameter, and for at least one member.

Arguments

u_cache	Constraint cache in which the *Matched Parameters* constraint must be created. See also u_cache,
l_instances	List of instance members to be matched for the value of the parameter registered as "`fingerWidth`".

Value Returned

`nil`	Always returns `nil`.

Example

If "`NM1`" and "`NM2`" are the names of two instances, the following function will create a *Matched Parameter* constraint for the parameter that corresponds to the registered *fingerWidth* parameter, and the members of that new constraint are "`NM1`" and "`NM2`":

```
cache = ciCacheGet(geGetEditCellView()
ciMatchedFingerWidth(cache list( list("/NM1" 'inst) list("/NM2" 'inst)))
```

ciMatchedParametersForCurrent_Mirror

```
ciMatchedParametersForCurrent_Mirror(
     u_cache
     l_instances
     )
     => nil
```

Description

Used the *Circuit Prospector* assistant as a generator to set *Matched Parameters* constraints to the selected instance members.

The parameters set for matching are the ones registered as *length*, *fingerWidth*, *mFactor* and *fingerCount* with the function `ciMapParam`.

The ratio value for the constraints are set for the *Matched Parameters* constraints applied for *mFactor* and *fingerCount*.

Note: The *Matched Parameters* constraints are either created when no such constraint exists for at least one member, or the new members are added to the *Matched Parameters* constraint when it exists for the same parameter, and for at least one member.

Arguments

u_cache	Constraint cache in which the *Matched Parameters* constraints must be created. See also u_cache,
l_instances	List of instance members to be matched for the value of the parameters registered as *length*, *fingerWidth*, *mFactor*, and *fingerCount*.

Value Returned

`nil`	Always returns `nil`.

Example

If “`NM1`” and “`NM2`” are the names of two instances, the following function creates one *Matched Parameter* constraint per parameter that corresponds to the registered *length*, *fingerWidth*, *mFactor*, and *fingerCount* parameters, and the members of these new constraints are "`NM1`" and "`NM2`":

```
cache = ciCacheGet(geGetEditCellView()
ciMatchedParametersForCurrent_Mirror(cache list( list("/NM1" `inst) list("/NM2"
`inst)))
```

ciMatchedParamsForInstanceSymmetry

```
ciMatchedParamsForInstanceSymmetry(
    u_cache
    l_instances
    )
    => nil
```

Description

Used by the *Circuit Prospector* assistant as a generator to set *Matched Parameters* constraints to the selected pair of instance members.

The parameters set for matching are the ones registered as *length*, *width*, *mFactor*, *fingerCount*, *fingerWidth*, *resValue*, and *capValue*, with the function `ciMapParam` and when found on the first instance.

The parameters corresponding to registered parameters are grouped into the same matched parameter constraint and applied to both devices.

The *Matched Parameters* constraints are either created when no such constraint exists for at least one member, or the new members are added to the *Matched Parameters* constraint when it exists for the same parameter set, and for at least one member.

Arguments

u_cache	Constraint cache in which the *Matched Parameters* constraints must be created. See also u_cache,
l_instances	List of instance members to be matched for the value of the parameters registered as *length*, *width*, *mFactor*, *fingerCount*, *fingerWidth*, *resValue*, and *capValue*.

Value Returned

`nil`	Always returns `nil`.

Example

If “`NM1`” and “`NM2`” are the names of two instances registered as “`nfet`”, the following function creates a *Matched Parameter* constraint for the pair of parameters that correspond to the registered *length*, *width*, *mFactor*, *fingerCount*, *fingerWidth*, *resValue*, and *capValue* parameters, and the members of these new constraints are “`NM1`” and “`NM2`”:

```
cache = ciCacheGet(geGetEditCellView()
ciMatchedParamsForSameSizeInstances(cache list( list(“/NM1” `inst) list(“/NM2”
`inst)))
```

ciMatchedParamsForSameSizeInstances

```
ciMatchedParamsForSameSizeInstances(
     u_cache
     l_instances
     )
     => nil
```

Description

Used by the *Circuit Prospector* assistant as a generator to set *Matched Parameters* constraints to the selected instance members.

The parameters set for matching are the ones registered as *length*, *width*, *resValue*, or *capValue*, with the function `ciMapParam`, and when found on the first instance.

- The parameters corresponding to *length* and *width* are grouped into the same matched parameter constraint and applied to devices registered as "`nfet`" or "`pfet`".
- The parameter corresponding to *resValue* is applied to devices registered as "`passive`".
- The parameter corresponding to *capValue* is applied to devices registered as "`passive`".

Note: The Matched Parameters constraints are either created when no such constraint exists for at least one member, or the new members are added to the *Matched Parameters* constraint when it exists for the same parameter set and for at least one member.

Arguments

u_cache	Constraint cache in which the *Matched Parameters* constraints must be created. See also u_cache,
l_instances	List of instance members to be matched for the value of the parameters registered as *length*, *width*, *resValue*, or *capValue*.

Value Returned

`nil`	Always returns `nil`.

Example

If “`NM1`” and “`NM2`” are the names of two instances registered as “`nfet`”, the following function creates a *Matched Parameter* constraint for the pair of parameters that correspond to the registered *length* and *width* parameters, and the members of these new constraints are “`NM1`” and “`NM2`”:

```
cache = ciCacheGet(geGetEditCellView()
ciMatchedParamsForSameSizeInstances(cache list( list(“/NM1” `inst) list(“/NM2”
`inst)))
```

ciMergeParams

```
ciMergeParams(
     l_paramsl
     l_paramsToMerge
     [ ?overwrite b_overwrite ]
     )
     => l_mergedParams
```

Description

Merges two lists of constraint parameters into a single list. By default, if the parameter being merged already exists in the list of parameters, the parameter value will be overwritten with the new value. The optional overwrite parameter can be set to `nil` to prevent parameter value overwrites.

Arguments

l_params	The parameters that `paramsToMerge` will be merged into
l_paramsToMerge	The parameters to be merged into `params`.
`?overwrite` *b_overwrite*	Identify whether the parameter values should be overwritten or not. Default: `t`

Value Returned

l_mergedParams	A merged parameters list with is the result of merging `paramsToMerge` into `params`.

Example

- Merging with overwrites allowed:

```
ciMergeParams('(("one" 1) ("two" 2) ("three" 3)) '(("one" "ONE") ("nine" "999")))
 '(
  ("nine" "999")
  ("one" "ONE")
  ("two" 2)
  ("three" 3)
)
```

- Merging with overwrites not allowed:

```
ciMergeParams('(("one" 1) ("two" 2) ("three" 3)) '(("one" "ONE") ("nine"
"999")) ?overwrite nil)

'(
 ("nine" "999")
 ("one" 1)
 ("two" 2)
 ("three" 3)
)
```

ciModgenForSameCellSizeAndBulk

```
ciModgenForSameCellSizeAndBulk(
     u_cache
     l_instances
     )
     => l_modgen / nil
```

Description

Used by the *Circuit Prospector* assistant as a generator to create Modgen constraints on groups of devices that have the same size and bulk connection.

Arguments

u_cache	Constraint cache where the instances can be found from their full path names (See `cache`.).
l_instances	List of members that have the same size and bulk connection and are also selected in the *Circuit Prospector* browser.

Value Returned

l_modgen	A list made of the constraint Id for the new *Modgen* constraint when it is created.
`nil`	Command failed.

Example

If `NM1` and `NM2` are the names of two instances, that have the same size and bulk connection, the following functions create a *Modgen* constraint with `NM1` and `NM2` as members:

```
cache = ciCacheGet(geGetEditCellView()
modgen = ciModgenForSameCellSizeAndBulk(cache list( list("/NM1" inst)
list("/NM2" inst)))
```

ciMOSActiveLoadStructIterator

```
ciMOSActiveLoadStructIterator(
     d_cellViewID
     t_matchExpression
     )
     => l_structuredObjects
```

Description

Iterator for all MOS active load structures.

Arguments

d_cellviewID	The cellview to apply the iterator to.
t_matchExpression	The expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_structuredObjects	The list of objects that satisfy the given matched expression.

Example

```
cv = geGetEditCellView()
ciMOSActiveLoadStructIterator(cv "t")
```

ciMOSCascodedCurrentMirrorStructIterator

```
ciMOSCascodedCurrentMirrorStructIterator(
     d_cellViewID
     t_matchExpression
     )
     => l_structuredObjects
```

Description

Iterator for all MOS cascoded current mirror structures.

Arguments

d_cellviewID	The cellview to apply the iterator to.
t_matchExpression	The expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_structuredObjects	The list of objects that satisfy the given matched expression.

Example

```
cv = geGetEditCellView()
ciMOSCascodedCurrentMirrorStructIterator(cv "t")
```

ciMOSCascodedCurrentMirrorStructIterator2

```
ciMOSCascodedCurrentMirrorStructIterator2(
     d_cellViewID
     t_matchExpression
     )
     => l_structuredObjects
```

Description

Iterator for all MOS cascoded current mirror structures, with at least one of the leading pairs being diode connected.

Arguments

d_cellviewID	The cellview to apply the iterator to.
t_matchExpression	The expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_structuredObjects	The list of objects that satisfy the given matched expression.

Example

```
cv = geGetEditCellView()
ciMOSCascodedCurrentMirrorStructIterator2(cv "t")
```

ciMOSCascodeIterator

```
ciMOSCascodeIterator(
     d_cellViewID
     t_matchExpression
     )
     => l_structuredObjects
```

Description

Iterator for MOS cascoded structures.

Arguments

d_cellviewID	Database ID of the cellview to which the iterator is to be applied.
t_matchExpression	Expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_structuredObjects	List of objects that satisfy the given expression.

Example

```
cv = geGetEditCellView()
ciMOSCascodeIterator(cv "t")
```

ciMOSCommonGateStructIterator

```
ciMOSCommonGateStructIterator(
     d_cellViewID
     t_matchExpression
     )
     => l_structuredObjects
```

Description

Iterator for all MOS common gate structures in a design.

Arguments

d_cellviewID	The cellview to apply the iterator to.
t_matchExpression	The expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_structuredObjects	The list of objects that satisfy the given matched expression.

Example

```
cv = geGetEditCellView()
ciMOSCommonGateStructIterator(cv "t")
```

ciMOSCrossCoupledDifferentialPairStructIterator

```
ciMOSCrossCoupledDifferentialPairStructIterator(
     d_cellViewID
     t_matchExpression
     )
     => l_structuredObjects
```

Description

Iterator for a MOS cross-coupled differential pair structure in a design. A MOS cross-coupled differential pair is a pair of MOS transistors that have the GATE of one transistor connected to the DRAIN of the other. The SOURCE of both transistors are connected to each other. By default, the SOURCE of the transistors cannot be connected to a POWER SUPPLY. This behavior can be changed by setting the environment variable `crossCoupleDiffPairRequireSourceNotSupply` to `nil`. To revert to the default behavior, set the environment variable back to `t`.

Arguments

d_cellviewID	The cellview to apply the iterator to.
t_matchExpression	The expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_structuredObjects	The list of objects that satisfy the given matched expression.

Example

```
cv = geGetEditCellView()
ciMOSCrossCoupledDifferentialPairStructIterator(cv "t")
```

ciMOSCrossCoupledQuadStructIterator

```
ciMOSCrossCoupledQuadStructIterator(
     d_cellViewID
     t_matchExpression
     )
     => l_structuredObjects
```

Description

Iterator for all MOS cross coupled quad structures in a design.

Arguments

d_cellviewID	The cellview to apply the iterator to.
t_matchExpression	The expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_structuredObjects	The list of objects that satisfy the given matched expression.

Example

```
cv = geGetEditCellView()
ciMOSCrossCoupledQuadStructIterator(cv "t")
```

ciMOSCurrentMirrorStructIterator

```
ciMOSCurrentMirrorStructIterator(
     d_cellViewID
     t_matchExpression
     )
     => l_structuredObjects
```

Description

Iterator for all MOS current mirror structures.

Arguments

d_cellviewID	The cellview to apply the iterator to.
t_matchExpression	The expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_structuredObjects	The list of objects that satisfy the given matched expression.

Example

```
cv = geGetEditCellView()
ciMOSCurrentMirrorStructIterator(cv "t")
```

ciMOSDifferentialPairStructIterator

```
ciMOSDifferentialPairStructIterator(
     d_cellViewID
     t_matchExpression
     )
     => l_structuredObjects
```

Description

Iterator for MOS differential pair structures in a design.

Arguments

d_cellviewID	The cellview to apply the iterator to.
t_matchExpression	The expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_structuredObjects	The list of objects that satisfy the given matched expression.

Example

```
cv = geGetEditCellView()
ciMOSDifferentialPairStructIterator(cv "t")
```

ciMOSInverterStructIterator

```
ciMOSInverterStructIterator(
     d_cellViewID
     t_matchExpression
     )
     => l_structuredObjects
```

Description

Iterator for all MOS inverter structures in a design.

Arguments

d_cellviewID	The cellview to apply the iterator to.
t_matchExpression	The expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_structuredObjects	The list of objects that satisfy the given matched expression.

Example

```
cv = geGetEditCellView()
ciMOSInverterStructIterator(cv "t")
```

ciMOSParallelStructIterator

```
ciMOSParallelStructIterator(
     d_cellViewID
     t_matchExpression
     )
     => l_structuredObjects
```

Description

Iterator for all MOS parallel structures.

Arguments

d_cellviewID	The cellview to apply the iterator to.
t_matchExpression	The expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_structuredObjects	The list of objects that satisfy the given matched expression.

Example

```
cv = geGetEditCellView()
ciMOSParallelStructIterator(cv "t")
```

ciMOSTransmissionGateStructIterator

```
ciMOSTransmissionGateStructIterator(
     d_cellViewID
     t_matchExpression
     )
     => l_structuredObjects
```

Description

Iterator for all MOS transmission gate structures.

Arguments

d_cellviewID	The cellview to apply the iterator to.
t_matchExpression	The expression to be applied by the iterator to find the required items in the given cellview.

Value Returned

l_structuredObjects	The list of objects that satisfy the given matched expression.

Example

```
cv = geGetEditCellView()
ciMOSTransmissionGateStructIterator(cv "t")
```

ciNetIterator

```
ciNetIterator(
     d_cellView
     t_finderNetExpr
     )
     => l_netList
```

Description

Used by the *Circuit Prospector* assistant to iterate over all design nets, collecting them together into groups based on the result of evaluating the passed expression (`finderNetExpr`). All nets which have the same `finderNetExpr` result are grouped together. If the finder match net expression evaluates to `nil`, then the net is ignored.

Arguments

d_cellview	Design cellview containing nets to be iterated.
t_finderNetExpr	Finder match expression using the variable net to be used to iterate.

Value Returned

l_netList — Grouped net list returned, for example:

```
list(
    list( list(nil dbId db:0x13eab01b
    hierPath nil) list(nil dbId db:0x13eab020
    hierPath nil) ...) ;;; net group A
    list( list(nil dbId db:0x13eab031
    hierPath nil) ...) ;;; net group B
    ...
)
```

Example

The `ciNetIterator` function evaluates the finder match expression with the current design net assigned to a local variable named *net*. The *net* variable can then be referenced in the finder match expression, for example:

To group all nets according to the number of instance terminals they are connected to:

```
finderNetExpr = "length(net->instTerms)"
```

```
netGroups = ciNetIterator(cv finderNetExpr)
foreach(netGroup netGroups
foreach( net netGroup print(net->dbId~>name) printf("\n"))
)
"net3""net5"
"net2""net8""net7"
"net4"
```

ciNetNames

```
ciNetNames(t_netType)
     => l_netNames / nil
```

Description

Returns the list of net names registered for `t_netType`.

Arguments

t_netType	A net type.

Value Returned

l_netNames	A list of net names.
nil	If a list of net names not found.

Examples

Example 1

```
ciRegisterNetNames("Power" '("vdd" "vcc"))
```

```
ciNetNames("Power") ; ("vdd" "vcc")
```

The above example returns `vdd` and `vcc` net names that are registered for the `Power` net type.

```
ciNetNames("MyType") ; nil
```

The above example returns `nil` as a list of net names not found.

Example 2

The `ciNetIterator` function evaluates the finder match expression with the current design net assigned to a local variable named *net*. The *net* variable can then be referenced in the finder match expression, for example:

To group all nets according to the number of instance terminals they are connected to:

```
finderNetExpr = "length(net->instTerms)"
```

```
netGroups = ciNetIterator(cv finderNetExpr)
foreach(netGroup netGroups
foreach( net netGroup print(net->dbId~>name) printf("\n"))
)
"net3""net5"
"net2""net8""net7"
"net4"
```

ciNetOnTerm

```
ciNetOnTerm(
     d_inst
     t_termName
     )
     => d_net / nil
```

Description

Utility function used by the *Circuit Prospector* assistant finders to retrieve the design instance connected to a named instance terminal.

Arguments

d_inst	The design instance.
t_termName	The instance terminal name.

Value Returned

d_net	Database ID of the net attached to an instance terminal.
nil	No design net found.

Example

```
desNet = ciNetOnTerm(desInst "G")
print(desNet->name)
"r4"
```

For extended example of use see example section for ciGetDeviceTermName.

ciNetPredicates

```
ciNetPredicates(
     t_netType
     )
     => l_predicates / g_predicate / nil
```

Description

Returns the net predicates registered for *t_netType*.

Arguments

t_netType	A net type.

Value Returned

l_predicates	A list of unique predicate functions from the sub-types.
g_predicates	A single net predicate for `t_netType`.
`nil`	if `t_netType` has no registered predicates.

Example

```
;; Get a single net predicate
ciRegisterNetPredicate("Ground" 'ciSigTypeMatchesNetType)
ciNetPredicates("Ground") ; ciSigTypeMatchesNetType

;; Get several predicates from a super-type.
procedure(myPredicate1(net netType) net->name == "xxx")
procedure(myPredicate2(net netType) net->name == "yyy")
ciRegisterNetPredicate("MyType1" 'myPredicate1)
ciRegisterNetPredicate("MyType2" 'myPredicate2)
ciRegisterNetSuperType("MySuperType" '("MyType1" "MyType2"))
ciNetPredicates("MySuperType") ; (myPredicate1 myPredicate2)
```

ciNetRegexs

```
ciNetRegexs(
     t_netType
     )
     => l_regexs / nil
```

Description

Returns the list of regular expressions registered for `t_netType`.

Arguments

t_netType	A net type.

Value Returned

l_regexs	A list of regular expressions.
`nil`	if no regular expressions are found.

Example

```
ciRegisterNetRegexs("Power" '("[vV][dD][dD]" "[vV][cC][cC]"))
ciNetRegexs("Power") ; ("[vV][dD][dD]" "[vV][cC][cC]")
ciNetRegexs("MyType"); nil
```

ciNextConName

```
ciNextConName(
     g_cache
     t_prefix
     )
     => t_conName
```

Description

Returns the next unique constraint name based on the given constraint name prefix. This function assumes that the constraints have names of the format `<prefix><number>`.

Arguments

g_cache	The constraints cache.
t_prefix	The constraint name prefix.

Value Returned

t_conName	Returns a unique constraint name of the format `<prefix><number>`.

Example

Assume that the cache does not contain any constraints prefixed with `Symm`, then:

```
symm1 = ciConCreate(cache 'symmetry ?name ciNextConName(cache "Symm") ?members
'(("NM1" inst) ("NM2" inst)))
symm2 = ciConCreate(cache 'symmetry ?name ciNextConName(cache "Symm") ?members
'(("NM3" inst) ("NM4" inst)))
symm3 = ciConCreate(cache 'symmetry ?name ciNextConName(cache "Symm") ?members
'(("NM5" inst) ("NM6" inst)))

symm1~>name
 "Symm"

symm2~>name
 "Symm1"

symm3~>name
```

```
"Symm2"
```

Related Functions

- ciNextObjName
- ciNextTemplateName

ciNextObjName

```
ciNextObjName(
     l_objectNames
     [ ?baseName t_baseName ]
     )
     => t_objName
```

Description

Returns the next unique object name based on the given base object name prefix and a list of existing object names. This function assumes the objects have names of the format *<baseName><number>*.

Arguments

l_objectNames	A list of existing object names.
?baseName *t_baseName*	The base object name. Default: *New*.

Value Returned

t_objName	Returns a unique object name of the format *<baseName><number>*.

Example

Assume that there are no constraints prefixed with the base name `Symm`, then:

```
ciNextObjName(cache->constraints~>name ?baseName "Symm")
   => "Symm"
ciNextObjName(append1(cache->constraints~>name "Symm") ?baseName "Symm")
   => "Symm1"
ciNextObjName(append(cache->constraints~>name list("Symm" "Symm1")) ?baseName
"Symm")
   => "Symm2"
```

Related Functions

- ciNextConName
- ciNextTemplateName

ciNextTemplateName

```
ciNextTemplateName(
     g_cache
     t_prefix
     )
     => t_templateName
```

Description

Returns the next unique template name based on the given template name prefix. This function assumes the templates have names of the format *<prefix><number>*.

Arguments

g_cache	The constraints cache.
t_prefix	The template name prefix.

Value Returned

t_templateName	Returns a unique template name of the format *<prefix><number>*.

Example

Assume that the cache does not contain any templates already prefixed with `Match`, then:

```
match1= ciTemplateCreate(cache "match" ?name ciNextTemplateName(cache "Match")
?members '(("PM6" inst) ("PM3" inst)) ?userParams 1)

match2= ciTemplateCreate(cache "match" ?name ciNextTemplateName(cache "Match")
?members '(("PM4" inst) ("PM5" inst)) ?userParams 2)

match3= ciTemplateCreate(cache "match" ?name ciNextTemplateName(cache "Match")
?members '(("PM1" inst) ("PM2" inst)) ?userParams 3)


match1~>name

"Match"


match2~>name

"Match1"


match3~>name
```

```
"Match2"
```

Related Functions

- ciNextConName
- ciNextObjName

ciNumTermsEQ2

```
ciNumTermsEQ2(
     d_net
     )
     => t / nil
```

Description

Utility function used by the *Circuit Prospector* assistant finders to check if the passed design net Id is connected to two terminals.

Arguments

d_net	The design net Id.

Value Returned

`t`	Design net connects to two terminals.
`nil`	Design net does not connect to two terminals.

Example

```
desNet = dbFindNetByName(geGetEditCellView() "r4")
ciNumTermsEQ2(desNet)
t
```

ciOrientationForModgen

```
ciOrientationForModgen(
     u_cache
     l_instances
     [ ?restrictTo t_orientation ]
     )
     => l_orientation / nil
```

Description

Used by the *Circuit Prospector* assistant as a generator to set an *Orientation* constraint to an existing *Modgen* constraint.

Note: The *Orientation* constraint is created only if a *Modgen* constraint exists for the given members.

Arguments

u_cache	Constraint `cache` that contains the *Modgen* constraint and in which the *Orientation* constraint must be created.
l_instances	List of members of the same *Modgen* constraint that are selected in the *Circuit Prospector* assistant's browser.
`?restrictTo` *t_orientation*	Name of the orientation value to be set. The value can be any of the following: `"R0"`, `"R90"`, `"R180"`, `"R270"`, `"MX"`, `"MY"`, `"MXR90"`, or `"MYR90"`. Default: `"R0"`

Value Returned

l_orientation	A list made of the constraint ID for the new *Orientation* constraint when it is created.
`nil`	No actions completed.

Example

If `NM1` and `NM2` are the names of two instances that are members of an existing *Modgen* constraint, the following functions creates an *Orientation* constraint and the member of that new constraint is the *Modgen* constraint for `NM1` and `NM2`:

```
cache = ciCacheGet(geGetEditCellView()
ciOrientationForModgen(cache list( list("/NM1" inst) list("/NM2" inst)))
```

ciParallelNetResistorArrayIterator

```
ciParallelNetResistorArrayIterator(
     d_cellview
     t_matchExpr
     )
     => l_returnedInsts / nil
```

Description

Iterates over all resistor devices, collating them into groups of parallelly-arranged resistors that start and end on the same net. This function is used by the *Parallel Net Resistor Array* finder of the Circuit Prospector assistant.

Arguments

d_cellview	The cellview containing the design instances to be iterated over.
t_matchExpr	The matched expression string to be used in the iteration.

Value Returned

l_returnedInsts	A list of returned instances. For example: `list(list( inst1 inst2 inst3 ...))`
nil	The device was ignored.

Example

For the *Parallel Net Resistor Array* finder,

```
matchExpr = "ciIsDevice(device \"resistor\")"
resDevices = ciParallelNetResistorArrayIterator(geGetEditCellView() matchExpr)
print(resDevices~>name
```

ciParallelResistorArrayIterator

```
ciParallelResistorArrayIterator(
     d_cellview
     t_matchExpr
     )
     => l_returnedInsts / nil
```

Description

Iterates over all resistor devices, collating them into groups of parallelly-arranged resistors that are all of the same length when their m-factor and iteration are expanded. This function is used by the *Parallel Resistor Array (Length)* finder of the Circuit Prospector assistant.

Arguments

d_cellview	The cellview containing the design instances to be iterated over.
t_matchExpr	The matched expression string to be used in the iteration.

Value Returned

l_returnedInsts	A list of returned instances. For example: `list(list( inst1 inst2 inst3 ...))`
`nil`	The device was ignored.

Example

For the *Parallel Resistor Array (Length)* finder,

```
matchExpr = "ciIsDevice(device \"resistor\")"
resDevices = ciParallelResistorArrayIterator(geGetEditCellView() matchExpr)
print(resDevices~>name)
```

ciPinIterator

```
ciPinIterator(
     d_cellView
     t_finderPinExpr
     )
     => l_pinList
```

Description

Used by the *Circuit Prospector* assistant finders to iterate over all design pins, collecting them together into groups based on the result of evaluating the passed finder expression (`finderPinExpr`). All pins which have the same `finderPinExpr` result are grouped together. If the `finderPinExpr` evaluates to `nil`, then the pin is ignored.

Arguments

d_cellView	Cellview containing design pins to be iterated.
t_finderPinExpr	Finder expression using the pin variable to iterate and group the pins with matching results.

Value Returned

l_pinList Grouped pin list returned, for example:

```
list(
    list( pinA1_pin pinA2_pin ...) ;;; pin group A
    list( pinB1_pin pinNameB2_pin ...) ;;; pin group B
    ...
)
```

Example

The `ciPinIterator` function evaluates the `finderPinExpr` with the current design pin assigned to a local variable can then be referenced in the `finderPinExpr`, for example:

To group all pins according to their `cellName`:

```
finderPinExpr = “pin->cellName”
pinGroups = ciPinIterator(cv finderPinExpr)
```

```
pinGroup1 = car(pinGroups)
print(car(pinGroup1~>cellName))
“ipin”
print(pinGroup1~>name)
(“in1” “in2” “in3”)
```

ciPlacerControlledWellGeneration

```
ciPlacerControlledWellGeneration(
     u_cache
     l_instsNetsPins
     [ ?wellType t_wellType ]
     )
     => nil
```

Description

Used by the Circuit Prospector assistant as a generator to set a placerControlledWell property for each given instance that is registered as a pfet.

The value of the *placerControlledWell* property is a pair-list with the name of the net connected to the bulk terminal for the given instance, and the layer name used for the well to be created for these instances.

Note: When the property is set, the Analog Placer will automatically create a rectangle with that layer below the respective instances.

Arguments

u_cache	Constraint cache where the instances can be found from their full path names (See cache.).
l_instsNetsPins	List of members (instances, nets, and pins) with their full path names and type, as selected in the *Circuit Prospector* browser.
?wellType *t_wellType*	Name of the layer used for the well of the given instances. By default, the name of the well layer is the layer defined with the function “nwell” in the technology file concatenated with the substring “Def”.

Value Returned

nil	Always returns nil.

Example

If `MP3` and `MP4` are the names of two instances that are registered as “`pfet`”, with the terminal registered as “`bulk`” connected to “`VDD`”, the following functions create a *placerControlledWell* property for “`MP3`” and “`MP4`”, with the value of that property being `("VDD" "NWELL")`:

```
cache = ciCacheGet(geGetEditCellView()
ciPlacerControlledWellGeneration(cache list( list("/MP3" `inst) list("/MP4"
`inst)) ?wellType "NWELL")
```

ciPrintMappedDefaultNetNames

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

Description

Returns the default net name registered with a particular terminal. Also returns details of the corresponding lib/cell/view.

Arguments

None

Value Returned

`t`	Command successful.
`nil`	Command failed.

ciPrintMappedDeviceNames

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

Description

Prints all default device names (for fets, BJTs, and so on) that have been registered using ciRegisterDevice.

Arguments

None

Value Returned

`t`	Print successful.
`nil`	Print unsuccessful.

Example

If a device has been registered as follows:

```
ciRegisterDevice("fet"
'('(nil "nmos" nil)
'(nil "pmos" nil)
'(nil "nmos3" nil)
'(nil "pmos3" nil)
'(nil "nmos4" nil)
'(nil "pmos4" nil)))
ciRegisterDevice("bjt"
'('(nil "npn" nil)
'(nil "pnp" nil)
'(nil "bjt504tnpn" nil)
'(nil "bjt504tpnp" nil)))
```

`ciPrintMappedDeviceNames()` prints device registration:

```
ciPrintMappedDeviceNames()
--------------------Device= "fet" -----------------------
Mapping : ('(nil "nmos" nil) '(nil "pmos" nil) '(nil "nmos3" nil) '(nil "pmos3"
nil) '(nil "nmos4" nil) '(nil "pmos4" nil))
--------------------Device= "bjt" -----------------------
Mapping : ('(nil "npn" nil) '(nil "pnp" nil) '(nil "bjt504tnpn" nil) '(nil
"bjt504tpnp" nil))
```

ciPrintMappedNetNames

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

Description

Prints all net categories registered using ciRegisterNet.

Arguments

None

Value Returned

`t`	Print successful.
`nil`	Print unsuccessful.

Example

```
ciRegisterNet("supply" '("vcc" "vdd" "vss"))
ciRegisterNet("ground" '("gnd!" "GND!"))
ciPrintMappedNetNames
-------------------Net= "supply" -----
Mapping : ("vcc" "vdd" "vss")
-------------------Net= "ground" ----
Mapping : ("gnd!" "GND!")
```

ciPrintMappedParams

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

Description

Prints all the parameters mapped by ciMapParam.

Arguments

None

Value Returned

`t`	Print successful.
`nil`	Print unsuccessful.

Example

If, for example, a mapping for `fetLength` is:

```
ciMapParam("fetLength" '("l"))
```

`ciPrintMappedParams()` will output the following mapping:

```
---------------Parameter= "fetLength" ----
Mapping : ("l")
```

Other default mappings for various parameters:

```
ciMapParam("length" '("l"))
ciMapParam("width" '("w"))
ciMapParam("fetlength" '("l","len","length"))
ciMapParam("fetwidth" '("w","wid","width"))
ciMapParam("fingerName"
'("fingers","finger","nfing","ng","fold","stripes","gates","gate","mi","mfactor",
"n"))
ciMapParam("wFinger" '("wFinger","w","G_W","effW","wg","wnom","fw"))
ciMapTerm("resValue" '("r"));
ciMapTerm("indValue" '("I"));
ciMapTerm("capValue" '("c"));
```

Here using the same API to store pwr/gnd names:

```
ciMapParam("pwrgnd"  '("vdd" "VDD" "vss" "VSS" "vcc" "VCC" "gnd" "GND"))         ;
"pwr/gnd"
ciMapParam("batt"  '("vbatt","VBATT"))          ;
ciMapParam("supply"
'("gnd","gnd!","GND","GND!","vss","vss!","VSS","VSS!","vssa","vssa!","VSSA","VSSA
!","vcc","vcc!","VCC","VCC!","vdd","vdd!","VDD","VDD!","vdda","vdda!","VDDA","VDD
A!"))
```

The respective `ciPrintMappedParams()` outputs would be:

```
--------------------Parameter= "length" -----------------------
Mapping : ("l")
--------------------Parameter= "width" ------------------------
Mapping : ("w")
--------------------Parameter= "fetlength" --------------------
Mapping : ("l" "len" "length")
--------------------Parameter= "fetwidth" ---------------------
Mapping : ("w" "wid" "width")
--------------------Parameter= "fingerName" -------------------
Mapping : ("fingers" "finger" "nfing" "ng" "fold" "stripes" "gates" "gate" "mi"
"mfactor" "n")
--------------------Parameter= "wFinger" ----------------------
Mapping : ("wFinger" "w" "G_W" "effW" "wg" "wnom" "fw")
--------------------Parameter= "resValue" -----------------------
Mapping : ("r")
--------------------Parameter= "indValue" -----------------------
Mapping : ("I")
--------------------Parameter= "capValue" -----------------------
Mapping : ("c")
--------------------Parameter= "pwrgnd" -----------------------
Mapping : ("vdd" "VDD" "vss" "VSS" "vcc" "VCC" "gnd" "GND")
--------------------Parameter= "batt" -------------------------
Mapping : ("vbatt" "VBATT")
--------------------Parameter= "supply" -----------------------
Mapping : ("gnd" "gnd!" "GND" "GND!" "vss" "vss!" "VSS" "VSS!" "vssa" "vssa!" "VSSA"
"VSSA!" "vcc" "vcc!" "VCC" "VCC!" "vdd" "vdd!" "VDD" "VDD!" "vdda" "vdda!" "VDDA"
"VDDA!")
```

ciPrintMappedTerminals

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

Description

Prints all the parameters mapped by ciMapTerm.

Arguments

None

Value Returned

`t`	Print successful.
`nil`	Print unsuccessful.

Example

If, for example, the mapping of terminal names is set as:

```
ciMapTerm("myGate" '("gate"))
```

`ciPrintMappedTerminals()` will print a mapping of:

```
-----------Terminal= "myGate" ----
 Mapping : ("G")
```

Other examples of default terminal name mappings:

```
ciMapTerm("drain" '("d" "D" "drain" "DRAIN" "Drain"))
ciMapTerm("gate" '("g" "G" "gate" "GATE" "Gate"))
ciMapTerm("source" '("s" "S" "source" "SOURCE" "Source"))
ciMapTerm("bulk" '("S" "BULK" "well" "SUB" "B"))
ciMapTerm("collector" '("c" "C" "collector"))
ciMapTerm("base" '("b" "B" "base" "BASE" "Base"))
ciMapTerm("emitter" '("e" "E" "emitter"))
ciMapTerm("resValue" '("r"))
ciMapTerm("indValue" '("I"))
ciMapTerm("capValue" '("c"))
```

The respective `ciPrintMappedTerminals()` outputs would be:

```
-------------------Terminal= "drain" ----------------------
Mapping : ("d" "drain" "DRAIN" "Drain")
-------------------Terminal= "gate" -----------------------
Mapping : ("g" "gate" "GATE" "Gate")
-------------------Terminal= "source" ---------------------
Mapping : ("s" "source" "SOURCE" "Source")
-------------------Terminal= "bulk" -----------------------
Mapping : ("B" "S" "BULK" "well" "SUB")
-------------------Terminal= "collector" ------------------
Mapping : ("c" "C" "collector")
-------------------Terminal= "base" -----------------------
Mapping : ("b" "B" "base" "BASE" "Base")
-------------------Terminal= "emitter" --------------------
Mapping : ("e" "E" "emitter")
```

ciRegisterAction

```
ciRegisterAction(
     l_argList
     )
     => t / nil
```

Description

Registers a list of attributes associated with an action and optionally adds the action to the constraints-related context menu, the Constraint Manager assistant's *Constraint Menu*, or both. This function is an alias for the `ciRegisterConstraintGenerator` function. The purpose of this alias is to emphasize the fact that constraint generator SKILL expressions can do everything possible in SKILL and not just create constraints.

See also ciRegisterConstraintGenerator, ciGetGenerator, and ciGetAction.

Arguments

l_argList	A list of attributes for an action. See `ciRegisterConstraintGenerator` `'args` <*l_args*> description for details.

Value Returned

`t`	Action was successfully registered.
`nil`	Action was not registered.

Example

This example registers an action, *List Constraints*, and adds this to the Constraint Manager assistant's *Constraint Menu*. The action prints out the names and/or types of the constraints in the current cellview. When selected from the *Constraint Menu*, a dialog box will open allowing you to choose whether the constraint types and/or names should printed out in the CIW.

```
configXMLstring = "<ConstraintConfig>
    <ConstraintType>
        <Name>List Constraints</Name>
        <MinMembers>0</MinMembers>
        <MaxMembers>10000000</MaxMembers>
```

```
        <AllowedMemberTypes>inst</AllowedMemberTypes>
        <IgnoreConstraintType>true</IgnoreConstraintType>
    </ConstraintType>
</ConstraintConfig>"

loadedOK = ciLoadConfigXMLFromString(configXMLstring)

printf("Load config.xml: %L\n" loadedOK)

ciRegisterAction(
    list(nil
        'name "List Constraints"
        'description "List the constraints in the CIW"
        'expression "when(args->\"List Types\" printf(\"%L\" cache-
        >constraints~>type)) when(args->\"List Names\" printf(\"%L\" cache-
        >constraints~>name))"
        'addToToolbar t
        'iconName "listConstraints"
        'args list(
        list("List Types" ‘bool t)
        list("List Names" ‘bool t)
        );args
    );list
);ciRegisterAction
```

ciRegisterAssistant

```
ciRegisterAssistant(
     l_disembodiedList
     )
     => t / nil
```

Description

Registers a new assistant (category) with the *Circuit Prospector* assistant. Assistants (categories) are groups of related *Circuit Prospector* finders. A category may represent a set of finders for a particular device type, for example fet, or a category could represent a top down/bottom up constraints flow. Categories can be registered automatically by placing them in a `.cadence/dfII/ci/categories` directory located on the Cadence File Search Path. The category registration SKILL code should be placed in a file in that directory with an `.il` extension, for example: `.cadence/dfII/ci/categories/myCategory.il`.

Arguments

l_disembodiedList	A disembodied list of objects to specify the category with the following objects: *name*: the name of the category. *description*: the description of the category (displayed as tooltip in Circuit Prospector). *finderNames*: the ordered list of finder names for the category.

Value Returned

`t`	Assistant/category successfully registered.
`nil`	Assistant/category not registered.

Example

```
ciRegisterAssistant( list(nil
    'name "Active Devices"
    'description "All active device finders"
    'finderNames list(
"Active Same Cell Name"
```

```
"Active Same Cell Name and Size"
"Active Common Gate"
"Active Same Well"
"Active X or Y Symmetric Pairs"
)
    )
)
```

ciRegisterConstraintGenerator

```
ciRegisterConstraintGenerator( list(nil
     'name              t_name
     'description       t_description
     'expression        t_expression
     [ 'addToToolbar    g_addToToolbar ]
     [ 'args            l_args ]
     [ 'callback        t_callbackExpr ]
     [ 'iconName        t_iconName ]
     [ 'menu            g_menu ]
     [ 'precondition    t_preconditionExpr ]
     [ 'settings        l_settings ]
     [ 'size            l_size ]
     [ 'title           t_title ]
     [ 'useCGenForEdit  g_useCGenForEdit ]
     )
     )
     => t / nil
```

Description

Registers a constraint generator with the *Constraint Manager* and, optionally, will make it available as an entry in the Constraint Creation toolbar drop-down list of the *Constraint Manager* and in the *Generate Constraints* list in the context-menu. The constraint generator can then be set as the default generator for a *Circuit Prospector* finder or used explicitly in the *Constraint Manager* to create constraints.

A constraint generator is a SKILL expression that, when evaluated, will create constraints. The expression is evaluated within an internal function that makes available variables for accessing the selected instances, nets, and pins (`instsNetsPins`) and the constraints cache. The constraint generator may optionally specify arguments which may cause a window to display when the generator is run. This is to allow you to specify values for these arguments. The values of the arguments may then determine what constraints are generated and their parameter settings. The argument values are made available to the constraint generator expression via a variable (args). The args variable being a disembodied property list and the argument values can be accessed through the argument name, for example `args->strength`. Argument types can be any one of the following: `bool`, `int`, `float`, `enum`, `string` (a single-line string), `multiString` (a multi-line string), `pattern` (a multi-line string with fixed-pitch font), `orient` (a multi-line string with fixed-pitch font), `separator`, `beginExpandedOptions`, or `endExpandedOptions`.

Note: All argument types can have a default value except `separator` and `endExpandedOptions` (see the **Examples** section).

See also ciUnregisterAssistant.

Arguments

`'name t_name` — The name of the constraint generator.

`'description t_description` — A description of the constraint generator.

`'expression t_expression` — A SKILL expression that when evaluated will generate constraints. The expression can refer to `instsNetsPins`, `cache`, and `args` variables.

`'addToToolbar g_addToToolbar`

Controls whether the constraint generator is added to the *Constraint Manager* toolbar.

`'args l_args` — The optional list of constraint generator arguments. Each argument should be specified in the following format:

```
<t_argName> <s_argType> [g_argDefVal]
```

Here, `<s_argType>` can have one of the following values: `'bool`, `'int`, `'float`, `'enum`, `'string` (a single-line string), `'multiString` (a multi-line string), `'pattern` (a multi-line string with fixed-pitch font), `'orient` (a multi-line string with fixed-pitch font), `'separator`, `'beginExpandedOptions`, or `'endExpandedOptions`.

Note: In the case of `'enum`, the remaining elements in the list are the possible `'enum` values.

The arguments can have an optional default value that can be of the same type as the argument type or a string that evaluates to a value of that type. For example,

```
("IntArg1" 'int    3)
("IntArg2" 'int   "2+3")
```

Additionally, it is possible to specify callbacks for each individual argument. The argument list specified by `'args` should be of the following format:

```
list( <t_argName> <s_argType>
[g_argDefVal] ['callback t_callBackExpr] )
```

A callback expression is called whenever the value of that particular argument is modified. When the callback expression is evaluated, the current values of all arguments are made available to the callback expression through an `args` variable, which is a disembodied property list of each argument name and its value. The callback expression can modify the values of `args` in the disembodied property list. These modified values will be displayed in the constraint generator dialog box. This feature can be used where there are dependencies between arguments, and updating one value leads to other argument values being updated.

Typically, a callback expression should be a call to a function of the following format:

```
exmplCallbackExampleArgCallback(
     cache argName args oldArgs instsNetsPins
     userEdit templateId callbackMode)
```

where:

- `cache`: The constraint cache.
- `args`: A disembodied property list of the current argument values displayed in the generator dialog.
- `argName`: The name of the argument that has been changed.
- `oldArgs`: A disembodied property list of the argument values in the dialog before the user made any changes. The `oldArgs` variable allows the callback to reset some or all of the constraint generator arguments back to the values they had before any edits were made.

- `instsNetsPins`: A list of lists where each sublist is the name of the selected instance, net, or pin.
- `userEdit`: A Boolean value set to `t` if you are making the change in the GUI, or `nil` if the constraint generator is being run for the first time and the dialog is being initialized with settings saved from the last time the dialog was used.
- `templateId`: The ID of the template that is being edited, or `nil` if the template has not been created yet.
- `callbackMode`: Passed to the callbacks set on the constraint generator arguments to identify the reason why the callback is called. This argument can have the following values:
    - `'userEdit` - value when the user modifies an argument using the GUI.
    - `'createNewTemplate` - value when a new template is created.
    - `'widgetProperties` - value when the constraint generator dialog box is loaded to edit an existing template if widget properties are enabled. For related information, see Enabling or Disabling Callbacks to Widget Properties section of the *Virtuoso Unified Custom Constraints Configuration Guide*.
    - `'templateParamChange` - value when external modification happens on the template parameters while the GUI is raised.
    - `'modgenTemplateCheck` - value when template parameters are set or updated on the template.

	Note: When `callbackMode == 'modgenTemplateCheck`, `args->widgetProperties` will return `nil`. Therefore, `widgetProperties` cannot be updated at this point.
	For an example of how to set the widget properties in the argument list, refer to the Setting Widget Properties in Constraint Generator Arguments section of the *Virtuoso Unified Custom Constraints Configuration Guide*.
	Some of these settings might not be appropriate and you can change the values in the callback as appropriate.
`'callback` *`t_callbackExpr`*	The callback expression is called whenever the *OK* button of the dialog box is clicked. The dialog box closes only if the callback expression returns `t`.
	The current values of all arguments are made available to the callback expression through an `args` variable, which is a disembodied property list of each argument name and its value. The current constraints cache and the list of instances, nets, and pins are also made available to the callback expression through the `cache` and `instsNetsPins` variables, respectively.
	For an example of how to write generator callbacks while defining widget properties in the argument list, refer to the Setting Widget Properties in Constraint Generator Arguments section of the *Virtuoso Unified Custom Constraints Configuration Guide*.
`'iconName` *`t_iconName`*	The name of the `.png` file of the icon to be used on the *Constraint Manager* toolbar.

`'menu g_menu`

An optional list of strings or a single string which will create hierarchical sub-menus attached to the top-level drop-down menu.

Where:

- A `menu` argument with a list of strings builds a hierarchical menu from left to right in the list of strings. For example,
  `'menu list("Custom1" "Custom2"`
- A `menu` argument as a single string creates a sub-menu on the top-level drop-down menu with all the constraints and constraint generators. For example,
  `'menu "Custom 1"`
- When no `menu` argument is specified, the constraint generator is placed on the top-level menu.

`'precondition t_preconditionExpr`

It is an optional argument that can be specified and must evaluate to `t` for the constraint generator to be run. If the constraint generator should not be run, the expression should evaluate to a string, which will be used in a message output to the log file and CIW. For example,

```
'precondition "if(ciWithinConstraint('modgen
cache ciGetMembersOfType(instsNetsPins
'inst)) then \"some or all of the devices are
already contained within an existing modgen\"
else t)"
```

You can use preconditions to check one of the following:

- Should the constraint generator icon in the Constraint Manager and the Circuit Prospector assistants be enabled or disabled.

- Whether the constraint generator needs to be run or not. If it needs to be run, then in which mode: Create (default behavior), Edit, or Replace.
- Whether an existing template matches the template that will be created or no matching template exists. This type of precondition is can be run only in Edit or Replace mode.

  If an existing template matches the one that will be created, then:

  - In Replace mode, the previously created template is deleted and the new one is created.
  - In Edit mode, the existing template loads in a dialog box for editing.
- Whether the template passes or fails. Based on this precondition, the template status is evaluated and displayed in the *Status* column of the Constraint Manager. For related information, refer to the Viewing Template Status in the Constraint Manager section in the *Virtuoso Unified Custom Constraints User Guide*.

Following is an example of the precondition:

```
procedure(precondition(checkType
instsNetsPins templateId)
let(((result nil))
    caseq(checkType
    ('EnableIcon
        ;; no templateId is passed when
        performing this check
        ;;available results:
        ;; if icon should be enabled
        ;; then result = t
        ;; else result != t
    )
```

```
    ('CreateEditReplace
        ;; no templateId is passed when
        performing this check
        ;;available results:
        ;; if template should be created then
        result = t
        ;; else if template should be edited
        then result = 'EditIfExists
        ;; else if template should be
        replaced then result =
        'ReplaceIfExists
        ;; else reason why the template
        cannot be created (existing behavior)
    )
    ('MatchTemplate
        ;; This the only time when templateId
        could be different than nil
        ;; If a precondition check for
        'CreateEditReplace returns either
        'EditIfExists or 'ReplaceIfExists,
        then precondition check to find the
        appropriate template is ran:
        ;; Based on the instsNetsPins if the
        template with the templateId is the
        one that should be edited or
        replaced.
        ;; If it should, then result = t
        ;; else, result = nil
    )
    ('ValidMembers
        result = t
        ;; Based on the result, the template
        status will be set.
        ;; If the generated constraint is no
        longer valid due to changes made in
        the schematic by the designer, then
        result = nil
        ;; else, result = t
    )

  )
  result
)
)
```

`'settings` *`l_settings`*	An optional disembodied property list of settings. This argument supports the following settings: ■ `'widgetPropertiesEnabled` Accepted Value: `nil` or `t` Setting `'widgetPropertiesEnabled` to `nil` disables the triggers to the callbacks for the status of widget properties. This reduces the number of calls to the callback when widget properties are not used. For related information, see Enabling or Disabling Callbacks to Widget Properties section of the *Virtuoso Unified Custom Constraints Configuration Guide*. ■ `'resizeMode` Accepted Value: `AutoResize` or `Manual` When `'resizeMode` is set to `AutoResize`, the constraint generator dialog box resizes automatically to the default minimum size for displaying all visible widgets. When `'resizeMode` is set to `Manual`, the size of the constraint generator dialog box on its first time access depends on the default minimum size for all visible widgets. However, you can thereafter increase or decrease the dialog box's size by dragging its border. Also, next time when you open the dialog box, it is not resized automatically; instead, it retains the dimensions you had set previously.
`'size` *`l_size`*	An optional list of two integers specifying the default width and height of the dialog box. If this argument is not specified, the dialog box is sized according to its contents.
`'title` *`t_title`*	An optional expression that should evaluate to string. The string will be used as the title for the dialog. If this argument is not specified, a default title is created based on the constraint generator name and the selected instances, nets, and pins.

`'useCGenForEdit` *`g_useCGenForEdit`*

By default, this argument is set to `nil`. When it is set to `t`, the constraint generator dialog box is displayed in the following two scenarios:

- When you double-click the template in the *Constraint browser*.
- When you click any template parameter in the *Constraint Parameter Editor* pane of the Constraint Manager assistant.

Note: This argument is set to `t` only for the generic module generators. Therefore, no other generators get affected and it is still possible to edit their parameters in the *Constraint Parameter Editor* pane of the Constraint Manager assistant.

Value Returned

`t`	Constraint generator successfully registered.
`nil`	Constraint generator not registered.

Examples

- Registering a constraint generator called “Matching (strength)”:

```
ciRegisterConstraintGenerator(
list(nil
    'name "Matching (strength)"
    'description "Generate various levels of Matching constraints"
    'expression "MyMatchingConstraintsGenerator(args instsNetsPins cache)" ;;;
    expression to generate constraints
    'addToToolbar t ;;; if you want to add a button to the toolbar for generator
    'iconName "templateMatched" ;;; icon to use on the toolbar.
    templateMatched.png must exist in the icon search path
    'args list( "strength" 'enum "low" "medium" "high")
    'menu list( "Custom1" "Custom2")
    'title "sprintf(nil \"Matching for %L\" mapcar(lambda((x) car(x))
    instsNetsPins))"
    'size list(100 100)
    )
```

```
)
```

- Setting default values for string, int, and float arguments:

```
ciRegisterConstraintGenerator(
list(nil
  'name         "My Template"
  'description  "My Template"
  'expression   "myTemplate(args instsNetsPins cache)"
  'addToToolbar t
  'iconName     "myTemplate"
  'args         list(
    list("myString" `string "abc")
    list("myIint"   `int    1234 )
    list("myFloat"  `float  12.34)
  );args
);list
);ciRegisterConstraintGenerator
```

- Generating the Current Mirror-specific modgen:

```
ciRegisterConstraintGenerator(
    list(nil
    'name "Module (Current Mirror)"
    'description "Generate Modgen Constraint for a Current Mirror"
    'expression "ciCreateModgen(cache instsNetsPins 'CurrentMirror args
    ?createModgenTemplate t)"
    'addToToolbar t
    'iconName "CurrentMirror"
    'args "ciGetStructArgs(`CurrentMirror)"
    'precondition "if(ciWithinConstraint('modgen cache
    ciGetMembersOfType(instsNetsPins 'inst)) then \"some or all of the devices
    are already contained within an existing modgen\" else t)".
)
```

- Enabling manual resizing of constraint generator dialog box:

```
ciRegisterConstraintGenerator(list(nil
    'name "MyGenericModgen"
    ...
    'settings list(nil 'widgetPropertiesEnabled t 'resizeMode "Manual")
    )
)
```

ciRegisterDefaultNetName

```
ciRegisterDefaultNetName(
     t_deviceCategory
     l_lcvNames
     t_terminalCategory
     t_termName
     t_defaultNetName
     )
     => t / nil
```

Description

Registers a default net name for the given terminal for either: the cellview of the same category or all lib/cell/view specified.

The registration function is used by those commands that automatically create wire stubs and wire names in the Virtuoso Schematic Editor.

The command will set the wire name according to the default net name, when a wire stub exists, or is created for the given cellview and terminal.

The default net name can be defined in two ways: registered for a device category and a terminal category, or for a list of L/C/V names and a terminal name.

Note: Both methods require the use of `t_defaultNetName`.

Arguments

t_devicecategory	The name of the device category.
l_lcvNames	A list of triplets for library, cell and view names.
t_terminalCategory	The name of the terminal category.
t_termName	The terminal name.
t_defaultNetName	The default net name.

Value Returned

`t`	Default net name successfully registered.
`nil`	Action failed.

Example

For a particular cellview and terminal name:

```
ciRegisterDefaultNetName('('("analogLib" "pmos" "symbol")) "B" "vdd!")
```

For a device category and a terminal category:

```
ciRegisterDefaultNetName("pfet" "bulk" "VDD")
```

ciRegisterDevice

```
ciRegisterDevice(
    t_deviceName
    l_deviceNameMapList
    )
    => t / nil
```

Description

Function used by the *Circuit Prospector* assistant to register a list of fet, nfet, pfet, BJT and passive device names to be used by `ciIsDevice`.

The `ciIsDevice` command is used to determine if an instance is a fet, nfet, pfet, BJT, passive device, or none of these. If `nil` is passed to `ciRegisterDevice`, the user-defined list is deleted and the internal default name list for the corresponding device type is used (see also `ciPrintMappedDeviceNames`).

All device type names starting with a lowercase letter are reserved for use by the *Circuit Prospector* and other Cadence functions. It is therefore recommended that you define them for the *Circuit Prospector* to find and operate on the correct devices.

The current list of reserved device types are:

- `fet`
- `nfet`
- `pfet`
- `bjt`
- `npn`
- `pnp`
- `diode`
- `passive`
- `resistor`
- `capacitor`
- `inductor`
- `standardCell`

Note: You can also define additional names starting with an uppercase letter for a list of device types for your own customization.

Arguments

t_deviceName	The name of the device to be registered.
l_deviceNameMapList	The mapping list for the named device using library, cell, and view name. An optional match expression string can also be used to register a group of devices.

Value Returned

`t`	Assistant/category registered.
`nil`	Assistant/category not registered.

Example

To register `fet` or `BJT` devices you can perform the following mapping:

```
ciRegisterDevice("fet"
    '((nil "nmos" nil)
      (nil "pmos" nil)
      (nil "nmos3" nil)
      (nil "pmos3" nil)
      (nil "nmos4" nil)
      (nil "pmos4" nil)))

ciRegisterDevice("bjt"
    '((nil "npn" nil)
      (nil "pnp" nil)
      (nil "bjt504tnpn" nil)
      (nil "bjt504tpnp" nil)))
```

To register all `pfet` devices that begin with `pmos*`, you can perform the following mapping using the optional matched expression:

```
ciRegisterDevice("pfet" '((nil nil nil (nil matchExpr "rexMatchp(\"^pmos\" device-
>cellName)"))))
```

To register all `nfet` devices that begin with `nmos*`, you can perform the following mapping using the optional matched expression:

```
ciRegisterDevice("nfet" '((nil nil nil (nil matchExpr "rexMatchp(\"^nmos\" device->cellName)"))))
```

To register all `fet` devices as a combination of `nfet` and `pfet` devices, you can perform the following mapping:

```
ciRegisterDevice("fet" append(ciGetDeviceNames("nfet") ciGetDeviceNames("pfet")))
```

Note: `nil` can be used as a wildcard entry. For example, where libName = `nil`, cellName = `cell1`, and viewName = `view1`, all devices with cellName "`cell1`" will still be categorized as "`user_defined_name`" irrespective of the `libName`.

Certain user-defined names may have already been registered but you can overwrite them.

`ciPrintMappedDeviceNames` lists all the registrations:

```
ciPrintMappedDeviceNames()
--------------------Device= fet
 Mapping : ('(nil nmos nil) '(nil pmos nil) '(nil nmos3 nil) '(nil
pmos3 nil) '(nil nmos4 nil) '(nil pmos4 nil))
--------------------Device= bjt ---
 Mapping : ('(nil npn nil) '(nil pnp nil) '(nil bjt504tnpn nil) '(nil
bjt504tpnp nil))
```

ciRegisterDevicesForPDKCategory

```
ciRegisterDevicesForPDKCategory(
     t_libName
     t_categoryName
     t_deviceTypeName
     )
     => l_regdDevices / nil
```

Description

Calls the ciRegisterDevice function to register all cells of the `t_categoryName` PDK category in the `t_libName` library as a device of type, `t_deviceTypeName`. In addition, the `ciRegisterDevicesForPDKCategory` function returns the list of devices registered because of the current action. This list does not includes the devices that were registered with the given device type name previously.

Arguments

t_libName	The name of the library, such as `"gpdk045"`.
t_categoryName	The name of the category, such as `"resistors"`.
t_deviceTypeName	The device type name of the cells located under the specified category in the specified library.

Value Returned

l_regdDevices — A list of registered devices. For example,

```
(
    ("t_categoryName" "cellName1" nil)
    ("t_categoryName" "cellName2" nil)
    ("t_categoryName" "cellName3" nil)
)
```

`nil` — Failed to find registered devices.

Example

The following function registers all cells under the category `"moscap"` in the library `"gpdk045"` as devices of type `"capacitor"`:

```
ciRegisterDevicesForPDKCategory("gpdk045" "moscap" "capacitor")
```

If the cells successfully get registered as `"capacitor"`, the function returns the following list:

```
(("gpdk045" "pmoscap2v" nil)
    ("gpdk045" "pmoscap1v" nil)
    ("gpdk045" "nmoscap2v" nil)
    ("gpdk045" "nmoscap1v" nil)
)
```

If cells have already been registered as `"capacitor"`, the function returns `nil`, as shown below.

```
ciRegisterDevicesForPDKCategory("gpdk045" "moscap" "capacitor")
nil
```

ciRegisterDynamicParamDef

```
ciRegisterDynamicParamDef(
     s_consTypeName
     t_paramTypeName
     l_paramDef
     )
     => t / nil
```

Description

Registers dynamic parameter definition for a constraint type.

Only a native constraint type can have a set of dynamic parameters in addition to the predefined (static) parameters defined in Virtuoso. Unlike a normal constraint parameter, a user specifies and registers a dynamic parameter definition at run time in SKILL. Dynamic parameter definitions for custom constraint types can be added by creating different definitions of the constraint using `config.xml`. These definitions can be dynamically loaded using the ciLoadConfigXML or ciLoadConfigXMLFromString function. However, ensure that you must not change the definition while a design is open using a different definition.

A set of dynamic parameters is predefined in Virtuoso for given constraint types, and you cannot change them. You are only allowed to specify dynamic parameter definitions using this SKILL function. A constraint type is said to be dynamic if it has at least one dynamic parameter definition. For example, a constraint type `A` has static parameters `P1` as integer and `P2` as a string. Constraint type `A` is allowed to have a dynamic parameter definition for parameter `P3`, which will be defined at run time. Then, use the following function to register the dynamic parameter definition:

```
ciRegisterDynamicParamDef('A "P3" '(parameter definition))
```

Arguments

s_consTypeName	The constraint type that owns the parameter's dynamic parameter definition. It should be a symbol, which represent a valid dynamic CI constraint that has some dynamic parameters allowed.
t_paramTypeName	The name of the dynamic parameter definition to be registered.

l_paramDef	List of strings, such as `method1`, `method2`, and `method3`, that define a dynamic parameter. A set of string values represents an enumeration that can be used in different ways depending on the type of the defined parameter. In case of an enumeration parameter, the first value in the list is considered as the default value of the dynamic parameter. For example, `method1` will be the default value for a dynamic parameter with the following enumeration: `method1`, `method2`, and `method3`.

Value Returned

`t`	If operation is sucessful.
`nil`	If operation is not sucessful.

Example

```
ciRegisterDynamicParamDef('matchedLDE "method" '("method1" "method2" "method3"))
```

This function defines dynamic parameter, `method`, for constraint Matched LDE Parameters, whose internal name is `matchedLDE` that can be accessed using the `ciListTypes` function. This is an enumeration parameter definition with the following values: `"method1" "method2" "method3"`. The definition specifies `"method1"` as a default value for the dynamic parameter.

ciRegisterFinder

```
ciRegisterFinder(
     t_name
     t_description
     t_iterator
     t_expression
     t_defaultCGen
     )
     => t / nil
```

Description

Registers a new finder with the *Circuit Prospector* assistant (although the recommended method is to do this using the Edit Finder form). Finders are used by the *Circuit Prospector* to iterate over a design and collect together groups of instances, nets, and pins that share common characteristics. The finder expression will determine how these objects are grouped together. Finders can be registered automatically by placing them in a `.cadence/dfII/ci/finders` directory located on the Cadence File Search Path. The finder registration SKILL code should be placed in a file in that directory with an `.il` extension, for example `.cadence/dfII/ci/finders/netTermCountFinder.il`.

Arguments

t_name	The name of the finder.
t_description	A detailed description of the finder.
t_iterator	The name of the iterator to use to find objects and apply the finder expression (see ciRegisterIterator).
t_expression	The finder expression to be applied by the iterator to group objects together. **Note:** All objects with the same finder expression are grouped together.
t_defaultCGen	The name of the default constraint generator associated with this finder. When the results appear in the *Circuit Prospector* results window, default constraints can be created by selecting the *Create Default Constraints* option in the *Constraint Manager* toolbar.

Value Returned

`t`	Finder successfully registered.
`nil`	Finder not registered.

Example

```
netTermCountFinder( list(nil
    'name "Net Term Counts"
    'description "Group nets according to terminal counts"
    'iterator "Net Iterator"
    'expression "length(net->instTerms)"
    'defaultCGen "IRDrop"
    )
)
```

ciRegisterIterator

```
ciRegisterIterator(
     t_name
     t_description
     t_iteratorFnName
     t_expression
     t_defaultCGen
     g_supportsFlattenedHier
     )
     => t / nil
```

Description

Registers a new iterator with the *Circuit Prospector* assistant (although the recommended method is to do this using the Edit Iterator form). Iterators are used by the *Circuit Prospector* finders to iterate over the design in a specific way, applying a finder expression which determines how objects are grouped.

All iterators should have two arguments: the cell view to be iterated over and the finder expression to be applied to the objects being iterated over. Iterators should return a list of grouped objects, a group being a list of object IDs. Grouping should be based on grouping together objects that share the same finder expression evaluation result. Objects should be ignored if the expression evaluates to `nil`. Since the finder expression is evaluated within the scope of the iterator function, it may contain references to any local variables that the iterator uses.

Iterators can be registered automatically by placing them in a `.cadence/dfII/ci/iterators` directory located in the Cadence File Search Path. The iterator registration SKILL code should be placed in a file in that directory with an `.il` extension, for example `.cadence/dfII/ci/iterators/netIterator.il`.

Arguments

t_name	The name of the iterator.
t_description	A detailed description of the iterator.
t_iteratorFnName	The name of the iterator function to use.
t_expression	The expression to be applied by the iterator to group objects together. **Note:** All objects with the same finder expression are grouped together.

t_defaultCGen	The name of the default constraint generator associated with this iterator. When the results appear in the *Circuit Prospector* results window, default constraints can be created by selecting the *Create Default Constraints* option in the *Constraint Manager* toolbar.
g_supportsFlattenedHier	If set, argument will get passed into `ciRunFinder` when a finder is run. If the finder does not support a flattened hierarchy, then the *Run finder on flattened hierarchy* option (in the *Run finder* pull-down of the *Circuit Prospector* assistant toolbar will be grayed out). Default setting is `nil`.

Value Returned

`t`	Iterator successfully registered.
`nil`	Iterator not registered.

Example

```
ciRegisterIterator(
  list(nil
    `name           "My Net Iterator"
    `description    "Iterate over all nets with expression vars: net, cellview"
    `iteratorFnName "myNetIterator"
  )
)

;;; Iterators must return a list of sub-lists where each sub-list represents a group
and is a list database IDs for the
;;; insts/nets/pins/instTerms in that group
;;;
procedure( myNetIterator(cellview finderExpr)
  let( (finderExprResults group groups result)

   ;;; Use an association table to group together objects with the same expression
evaluation result
    finderExprResults = makeTable("finderExprResults" nil)
```

```
    foreach(net cellview->nets
      ;;; Expression can refer to net, cellview
      result = evalstring(finderExpr)
      ;;; Group all non-nil results
      when(result finderExprResults[result] = append(finderExprResults[result]
list(net)))
    )

    ;;; Create a list of grouped objects from the association table
    groups = list()
    foreach(result finderExprResults
      group = finderExprResults[result]
      groups = append(groups list(group))
    )

    groups
  )
)
```

ciRegisterNet

```
ciRegisterNet(
     t_netType
     l_netNameList
     [ ?regexNetNames l_regExpressionList ]
     [ ?predicate g_predicate ]
     )
     => t / nil
```

Description

Registers the net names, regular expressions and predicates that make up a net type. A net belongs to `t_netType` if its name is one of `l_netNameList`, or if it matches one of `l_regExpressionList`, or if `g_predicate` returns `t`.

Note: User categories must start with an uppercase to avoid any overlap with the Cadence namespace used for pre-defined categories. The pre-defined categories are: `analog`, `clock`, `ground`, `power`, `reset`, `scan`, `supply`, `tieHi`, `tieLo`, and `tieOff`.

Note: The `?regexNetNames` and `?predicate` arguments default to `nil` if you do not explicitly specify them, resulting in *`no`* regular expressions or predicate. You may wish to use a more specific function to register net names, regular expressions or predicates individually.

Virtuoso ships with the following built-in net registrations:

```
ciRegisterNet( "power"     '("avdd" "avdd!" "vdd" "vdd!" "vdda" "vdda!"
                             "AVDD" "AVDD!" "VDD" "VDD!" "VDDA" "VDDA!"
                             "avcc" "avcc!" "vcc" "vcc!"
                             "AVCC" "AVCC!" "VCC" "VCC!")
                             ?regexNetNames list("[Vv][Dd][Dd]" "[Vv][Cc][Cc]")
                             ?predicate 'ciSigTypeMatchesNetType)
ciRegisterNet( "ground"    '("agnd" "agnd!" "gnd" "gnd!" "gnda" "gnda!"
                             "AGND" "AGND!" "GND" "GND!" "GNDA" "GNDA!"
                             "avss" "avss!" "vss" "vss!" "vssa" "vssa!"
                             "AVSS" "AVSS!" "VSS" "VSS!" "VSSA" "VSSA!")
                             ?regexNetNames list("[Gg][Nn][Dd]" "[Vv][Ss][Ss]")
                             ?predicate 'ciSigTypeMatchesNetType)
ciRegisterNetSuperType( "supply" '("power" "ground"))
```

See also:

- ciRegisterNetNames
- ciRegisterNetRegexs

- ciRegisterNetPredicate
- ciGetNetNames
- ciIsNet

Arguments

t_netType	A pre-defined or user-defined type in which to register nets.
l_netNameList	A list of net names that belong to `t_netType`. See also ciRegisterNetNames.
`?regexNetNames` *l_regExpressionList*	
	A list of regular expressions that catch the net names belonging to `t_netType`. See also ciRegisterNetRegexs.
`?predicate` *g_predicate*	A net predicate that determines whether or not a net belongs in `t_netType`. See also ciRegisterNetPredicate.

Value Returned

`t`	Net category successfully registered with specified net list.
`nil`	Net category not registered.

Example

```
;; Register net names, regular expressions, and predicate in one go.
ciRegisterNet("Power" '("vcc")
                      ?regexNetNames '("[vV][dD][dD]")
                      ?predicate 'ciSigTypeMatchesNetType)

ciNetNames("Power") ; ("vcc")
ciNetRegexs("Power") ; ("[vV][dD][dD]")
ciNetPredicates("Power") ; ciSigTypeMatchesNetType
```

ciRegisterNetNames

```
ciRegisterNetNames(
     t_netType
     l_netNames
     )
     => t / nil
```

Description

Registers a list of net names recognized by `ciIsNet()`.

Note: The ciRegisterNetRegexs and ciRegisterNetPredicate functions provide more general ways to assign several nets to a net type.

Arguments

t_netType	A net type.
t_netNames	A list of net names.

Value Returned

`t`	A list of net names successfully registered.
`nil`	List of net names not registered.

Example

```
;; Register a fixed list of "Power" nets.
ciRegisterNetNames("Power" '("vdd" "vcc"))

;; Classify some nets in a cellview.
cv  = geGetEditCellView()
vdd = dbFindNetByName(cv "vdd")
gnd = dbFindNetByName(cv "gnd")
VCC = dbFindNetByName(cv "VCC")

ciIsNet(vdd "Power") ; t
ciIsNet(gnd "Power") ; nil - "gnd" not a power net name.
ciIsNet(VCC "Power") ; nil - registered names are case-sensitive.
```

ciRegisterNetPredicate

```
ciRegisterNetPredicate(
     t_netType
     g_predicate
     )
     => t / nil
```

Description

Registers a net predicate function used by `ciIsNet()`.

A predicate is a SKILL function that takes a net and net type as arguments and returns `t` or `nil` to indicate whether or not the net belongs to the net type.

Arguments

t_netType	A net type.

g_predicate	A predicate function that either use a quoted function name, such as 'myPredicate, or a lambda(). The function must follow the following template:
	`procedure(myPredicate (net netType)`
	`;; return t if net belongs to netType; nil if not.`
	`)`

Value Returned

`t`	A net predicate function successfully registered.
`nil`	Operation failed.

Example

```
;; Register a fixed list of "Power" nets.
ciRegisterNetNames("Power" '("vdd" "vcc"))

;; Classify some nets in a cellview.
cv  = geGetEditCellView()
```

```
vdd = dbFindNetByName(cv "vdd")
gnd = dbFindNetByName(cv "gnd")
VCC = dbFindNetByName(cv "VCC")

ciIsNet(vdd "Power") ; t
ciIsNet(gnd "Power") ; nil - "gnd" not a power net name.
ciIsNet(VCC "Power") ; nil - registered names are case-sensitive.
```

ciRegisterNetRegexs

```
ciRegisterNetRegexs(
     t_netType
     l_Regexs
     )
     => t / nil
```

Description

Registers a list of regular expressions recognized by `ciIsNet()`.

A net belongs to `t_netType` if its name matches any of these expressions.

Arguments

t_netType	A net type.
l_Regexs	A list of regular expressions.

Value Returned

t	A list of regular expressions successfully registered.
nil	Operation failed.

Example

```
;; Register regular expressions matching anything containing 'vdd' or 'vcc'.
ciRegisterNetRegexs("Power" '("[vV][dD][dD]" "[vV][cC][cC]"))

cv      = geGetEditCellView()
vdd     = dbFindNetByName(cv "vdd")
avdd    = dbFindNetByName(cv "avdd")
Vdd     = dbFindNetByName(cv "Vdd")
vddbang = dbFindNetByName(cv "vdd!")
VCC     = dbFindNetByName(cv "VCC")

ciIsNet(vdd     "Power") ; t
ciIsNet(avdd    "Power") ; t
ciIsNet(Vdd     "Power") ; t
```

```
ciIsNet(vddbang "Power") ; t
ciIsNet(VCC     "Power") ; t
```

ciRegisterStructure

```
ciRegisterStructure(
     l_propertyList
     )
     => t / nil
```

Description

Registers a structure with the *Circuit Prospector* assistant (although the recommended method is to do this using Capturing New Structures).

Arguments

l_propertyList	List of properties to be applied to structure being registered.

Value Returned

`t`	Structure successfully registered.
`nil`	Structure not registered.

Example

```
ciRegisterStructure
    ( list(nil
        `name "MyStruct"
        `type "none"
        `description "none"
        `insts list(list(nil
        `instName "D0"
        `instId 0
        `expr "__ciInst__->name"
        `terms list(
    list(nil `name "MINUS" `net "vdd!")
    list(nil `name "PLUS" `net "vcom")
        )
    )
    list(nil
        `instName "MP3"
```

```
        'instId 1 'expr "__ciInst__->libName == 'gpdk446' && __ciInst__ >cellName
        == 'pmos3'"
        'terms list(
    list(nil 'name "S" 'net "vdd!")
    list(nil 'name "G" 'net "r4")
    list(nil 'name "D" 'net "r4")
        )
    )
    list(nil
        'instName "MP11"
        'instId 2
        'expr "__ciInst__->libName == 'gpdk446' && __ciInst__->cellName ==
        'pmos3'"
        'terms list(
    list(nil 'name "S" 'net "vdd!")
    list(nil 'name "G" 'net "r4")
    list(nil 'name "D" 'net "vdd!")
        )
    )
    list(nil
        'instName "MP10"
        'instId 3
        'expr "__ciInst__->libName == 'gpdk446' && __ciInst__->cellName ==
        'pmos3'"
        'terms list(
    list(nil 'name "S" 'net "vdd!")
    list(nil 'name "G" 'net "r2")
    list(nil 'name "D" 'net "n20")
        )
    )
    list(nil
        'instName "MP6"
        'instId 4
        'expr "__ciInst__->libName == 'gpdk446' && __ciInst__->cellName ==
        'pmos3'"
        'terms list(
    list(nil 'name "S" 'net "n20")
    list(nil 'name "G" 'net "r3")
    list(nil 'name "D" 'net "r2")
        )
    )
    list(nil
```

```
    'instName "MP5"
    'instId 5
    'expr "__ciInst__->libName == 'gpdk446' && __ciInst__->cellName ==
    'pmos3'"
    'terms list(
list(nil 'name "S" 'net "n17")
list(nil 'name "G" 'net "r3")
list(nil 'name "D" 'net "n18")
    )
)
list(nil
    'instName "MP7"
    'instId 6
    'expr "__ciInst__->libName == 'gpdk446' && __ciInst__->cellName ==
    'pmos3'"
    'terms list(
list(nil 'name "S" 'net "n18")
list(nil 'name "G" 'net "r3")
    )
)
list(nil
    'instName "MP9"
    'instId 7
    'expr "__ciInst__->libName == 'gpdk446' && __ciInst__->cellName ==
    'pmos3'"
    'terms list(
list(nil 'name "S" 'net "vdd!")
list(nil 'name "G" 'net "r3")
list(nil 'name "D" 'net "n17")
    )
)
list(nil
    'instName "MP2"
    'instId 8
    'expr "__ciInst__->libName == 'gpdk446' && __ciInst__->cellName ==
    'pmos3'"
    'terms list(
list(nil 'name "S" 'net "vdd!")
list(nil 'name "G" 'net "n3")
    )
)
list(nil
```

```
        `instName "D1"
        `instId 9
        `expr "__ciInst__->libName == 'gpdk446' && __ciInst__->cellName == 'pdio'"
        `terms list(
    list(nil `name "MINUS" `net "vdd!")
    list(nil `name "PLUS" `net "vcop")
        )
    )
    list(nil
        `instName "MP0"
        `instId 10
        `expr "__ciInst__->libName == 'gpdk446' && __ciInst__->cellName ==
        'pmos3'"
        `terms list(
    list(nil `name "S" `net "vdd!")
    list(nil `name "G" `net "n3")
        )
    )
    list(nil
        `instName "MP1"
        `instId 11
        `expr "__ciInst__->libName == 'gpdk446' && __ciInst__->cellName ==
        'pmos3'"
        `terms list(
    list(nil `name "S" `net "vdd!")
    list(nil `name "G" `net "n3")
    list(nil `name "D" `net "n3")
        )
    )
    list(nil
        `instName "MP4"
        `instId 12
        `expr "__ciInst__->libName == 'gpdk446' && __ciInst__->cellName ==
        'pmos3'"
        `terms list(
    list(nil `name "S" `net "vdd!")
    list(nil `name "G" `net "n2")
    list(nil `name "D" `net "n2")
        )
        )
        )
        `repeatableInsts nil
```

```
        'nets list(
    list(nil
        'name "vdd!"
        'expr "t"
    )
    list(nil
        'name "r4"
        'expr "t"
    )
    list(nil
        'name "n2"
        'expr "t"
    )
    list(nil
        'name "vcom"
        'expr "t"
    )
    list(nil
        'name "n3"
        'expr "t"
    )
    list(nil
        'name "vcop"
        'expr "t"
    )
    list(nil
        'name "n17"
        'expr "t"
    )
    list(nil
        'name "n18"
        'expr "t"
    )
    list(nil
        'name "r2"
        'expr "t"
    )
    list(nil
        'name "n20"
        'expr "t"
    )
```

```
list(nil
    'name "r3"
    'expr "t"
)
)
    'pins list(
)
    'matchExpr "t"
    'finder "MyStruct"
    'constraints list(
)
    )
     )
```

ciResolveNet

```
ciResolveNet(
     d_netDBid
     g_hierContext
     [ ?simplify t_simplify ]
     )
     => g_resolvedNetInfo / nil
```

Description

Resolves the passed net to the highest equivalent design net. Used by Circuit Prospector finders when run on flattened hierarchies. For example, the Active Common Gate finder.

Arguments

d_netDBid	A design net hierContext. A local variable that is defined when running a Circuit Prospector finder.
d_hierContext	The current Circuit Prospector hierarchical context.
?simplify *t_simplify*	If this option is specified, the resolved net returned by this function will be the simplest equivalent net. For example, if the resolved net is `<*6>net9` and a net named `net9` also exists then the function will return information based on `net9`.

Value Returned

g_resolvedNetInfo	The highest level equivalent design net or the passed net (if it is an internal net).
nil	Structure not registered.

Example

```
when(ciIsDevice(device \”fet\”)
ciResolveNet(ciNetOnTerm(device ciGetDeviceTermName(device \"gate\"))
hierContext)
```

Note: `hierContext` will only be valid when a finder is run on a flattened hierarchy (see *Run finder*...). In all other cases, `ciResolveNet` returns the same net that has been passed into it.

ciRunFinder

```
ciRunFinder(
     t_finderName
     t_iteratorFnName
     t_finderExpression
     t_constraintGenExpr
     [ g_iteratorSupportFlatHier ]
     [ t_hierScope ]
     [ n_depth ]
     [ g_cache ]
     [ s_predicate ]
     )
     => list
```

Description

Runs a *Circuit Prospector* finder and returns a list of objects grouped according to the finder expression evaluation result. If the `depth` argument is specified and the `hierScope` argument is `'depthCellViews`, the finder searches only up to the specified depth. If `cache` is specified, the finder runs on that cache; otherwise, the search results are obtained from the current window. See also ciRegisterFinder and ciRegisterIterator.

Arguments

t_finderName	The finder to be run.
t_iteratorFnName	The associated finder iterator function name.
t_finderExpression	The finder expression to be used.
t_constraintGenExpr	The constraint generator expression to be used.
g_iteratorSupportFlatHier	Set to `true` if iterator supports a flat hierarchy. The default is `nil`.
t_hierScope	The hierarchical scope, where, `hierScope = ( 'currentCellViewOnly | 'allCellViews | 'flattenedCellView | 'depthCellViews)`
n_depth	The depth up to which the finder should search in the hierarchy. This argument is considered only if the `hierScope` argument is set to `'depthCellViews`. Default value: `-1` (means all depth levels)

g_cache	The constraints cache.
s_predicate	Symbol of a function or a lambda function that accepts the following four arguments: `libName`, `cellName`, `viewName`, and `depth`. The `predicate` function should return `t` if the cellview should be included in the search list and `nil` otherwise. It can be used to reduce the time spent for running a structure finder. For example, it can be used to run finder for the *MOS Current Mirror* structure in the analog part of the design instead of the whole design. The `finderName` variable is also available to the predicate because of the closure.

Value Returned

list — List of finder run results.

For example:

```
list(
    list( objA1 ... objAN) ;;; object group A
    list( objB1 ... objBN ;;; object group B
    )
```

Examples

```
ciRunFinder( "Net Term Counts" "Net Iterator" "length(net->instTerms)" "" nil
'allCellViews -1 ciGetCellView())
```

The command above results in net database objects that are grouped according to the number of `instTerms` on the net, as shown below:

```
(
    ("net1" "net2" "net3")
    ("net99" "net100")
)
```

Note: The `depth` argument is currently ignored because `hierScope` is set to `'allCellViews`.

```
procedure(myPredicate(libName cellName viewName depth)
  libName ==  AnalogLib
)
```

```
ciRunFinder( "Net Term Counts" "Net Iterator" "length(net->instTerms)" "" nil
'allCellViews -1 ciGetCellView() 'myPredicate)
```

```
nil
```

The example above of using the `predicate` argument with `ciRunFinder` looks only into cellviews that are part of the library, `AnalogLib`.

ciRunFindersAndGenerators

```
ciRunFindersAndGenerators(
     g_cache
     t_categoryName
     [ ?runGenerators g_runGenerators ]
     [ ?deleteExisting g_deleteExisting ]
     [ ?addHierNotes g_addHierNotes ]
     [ ?printFinderResults g_printFinderResults ]
     [ ?updateArgsExpr g_updateArgsExpr ]
     [ ?triggerCBinMode g_triggerCBinMode ]
     )
     => list
```

Description

Runs all the finders and their corresponding generators for the specified constraint cache and *Circuit Prospector* category. The finders first find all the results and then the constraint generators are ran on those results to create constraints and templates.

Arguments

g_cache — The constraints cache.

t_categoryName — Name of the *Circuit Prospector* category.

?runGenerators *g_runGenerators*

When this argument is to set to `t`, the constraint generators are run; otherwise, only the finders are run.

Valid values: `t` and `nil`
Default value: `nil`

?deleteExisting *g_deleteExisting*

When this argument is to set to `t`, the existing templates and constraints in the cache are deleted before the finders and constraint generators are run.

Valid values: `t` and `nil`
Default value: `nil`

?addHierNotes *g_addHierNotes*

When this argument is to set to `t`, the notes are added for the constraints after the constraint generators are run.

Valid values: `t` and `nil`
Default value: `t`

`?printFinderResults` *`g_printFinderResults`*

When this argument is to set to `t`, the finder results are also displayed in the CIW.

Valid values: `t` and `nil`
Default value: `nil`

`?updateArgsExpr` *`g_updateArgsExpr`*

When this argument is to set to an expression, the string value is evaluated to get the arguments that need to be updated before evaluating the constraint generator expression. This expression can take a constraint generator name, `cGenName`, as an argument. `cGenName` is a string.

Valid values: `"expr"` and `nil`
Default value: `nil`

`?triggerCBinMode` *`g_triggerCBinMode`*

The mode in which callbacks are triggered for the argument values that need to be updated.

Valid values: `'userEdit` and `'createNewTemplate`
Default value: `'userEdit`

Note: Any other value does not triggers the callbacks for the time being. It has no effect if no arguments have been returned by the `"expr"` argument of `g_updateArgsExpr`.

Value Returned

list	A DPL containing the list of constraint generator expressions for each the constraint generator name is run under the specified constraint category.

Example

```
updateArgsExpression =  "let(((argsToUpdate nil)) when(cGenName == \"Module
(Current Mirror)\" argsToUpdate = list(list(\"Add GuardRing\" t)) ) argsToUpdate)"

ciRunFindersAndGenerators(ciGetCellView() "Rapid Analog Prototype"
?printFinderResults nil ?updateArgsExpr updateArgsExpression)

(nil Module\ \(Cascode\ MOS\ Transistors\) (ci:0x333c7090 ci:0x333e5f10) Module\
\(Cascoded\ Current\ Mirror\) (ci:0x3343a7c0)

Module\ \(Cascoded\ Current\ Mirror\) nil Module\ \(Current\ Mirror\)
(ci:0x3346ee80 ci:0x3348b960) Module\ \(Diff\ Pair\)

(ci:0x31ec6700) Module\ \(Diff\ Pair\) nil Module\ \(Passive\ Device\ Array\) nil

Module\ \(Large\ mfactor\) nil Symmetry\ \(default\ axis\) (ci:0x322bfee0 ci:0x0
ci:0x0 ci:0x322de9a0 ci:0x0

    ci:0x0

) Cluster\ for\ Standard\ Cells

nil Orientation\ Vertical (ci:0x334256f0 ci:0x2f5f6b00) Negative\ Supply\ Route\
Priority (

    ("VSS" net)

)

Positive\ Supply\ Route\ Priority (

    ("VDD" net)

) Symmetry\ \(default\ axis\) (ci:0x3314a5c0 ci:0x31ef5ce0 ci:0x334a0130
ci:0x31f37b60) Symmetry\ for\ Pins

(ci:0x33310ec0) Alignment\ for\ Top\ Pins (ci:0x0) Alignment\ for\ Bottom\ Pins
(ci:0x0)

Alignment\ for\ Left\ Pins (ci:0x33053a00) Alignment\ for\ Right\ Pins
(ci:0x325fece0) Enforce\ Modgen\ Symmetry\ Precedence

(t)

)
```

ciRunGenerator

```
ciRunGenerator(
     g_cache
     t_cGenName
     l_instsNetsPins
     [ ?argValuesToUpdate g_argValuesToUpdate ]
     [ ?triggerCBinMode g_triggerCBinMode ]
     )
     => list
```

Description

Given a constraint cache, a generator name and a list of objects(insts, nets, pins & terminals), the function Runs the specified constraint generator in the given constraint cache for the specified list of objects, that is, instances, nets, pins, and terminals. This function is called by ciRunFindersAndGenerators to run the constraint generator.

Arguments

g_cache	The constraints cache.
t_cGenName	Name of the constraint generator.
l_instsNetsPins	The list of instances, pins, nets, terminals, and so on. It is of the following format: `list(list("inst1" 'inst) list("inst2" 'inst) list("net1" 'net))`

`?argValuesToUpdate` *`g_argValuesToUpdate`*

When this argument is to set to `argValuesToUpdate`, a list of argument values that need to be updated after the first evaluation are specified. This list can be one of the following:

- A DPL starting with `nil`
- A DPL without `nil` as the first element of the list
- An association list between the argument and the value

Valid values: `"argValuesToUpdate"` and `nil`
Default value: `nil`

`?triggerCBinMode` *`g_triggerCBinMode`*

The mode in which callbacks are triggered for the argument values that need to be updated.

Valid values: `'userEdit` and `'createNewTemplate`
Default value: `'userEdit`

Note: Any other value does not triggers the callbacks for the time being. It has no effect if no arguments have been returned by the `"expr"` argument of `g_updateArgsExpr`.

Value Returned

`list`	The result when the generator expression is evaluated.

Example

The example below runs a *Module (Current Mirror)* generator on a particular set of instances and nets.

```
ciRunGenerator(ciGetCellView() list(list("/MP1" 'inst) list("/MP0" 'inst) list("/
MP2" 'inst) list("/n3" 'net) list("/vdd!" 'net)) "Module (Current Mirror)")
```

```
ciRunGenerator(ciGetCellView() "GenericModgen" list('("M1" inst) '("M2" inst)
'("INN" net)) ?triggerCBinMode 'userEdit ?argValuesToUpdate list('Num\ Rows 2))
ci:0x3262a7f0 ;; if successful returns the template created
```

```
ciRunGenerator(ciGetCellView() "GenericModgen" list('("M1" inst) '("M2" inst)
'("INN" net)) ?triggerCBinMode 'userEdit ?argValuesToUpdate list('Num\ Rows 2))
INFO (CMGR-6149): Running constraint generator 'GenericModgen'.
*WARNING* (CMGR-3125): Unable to run the constraint generator "GenericModgen" on
objects '( ("M1" inst) ("M2" inst) ("INN" net)) because some or all of the devices
are already contained within an existing modgen.
nil ;; failed to create the template
```

Note: `?argValuesToUpdate` can be specified as one of the following:

```
list(list("Num Rows" 2))
list(nil "Num Rows" 2)
list("Num Rows" 2)
list('Num\ Rows 2), ....
```

For the argument name, symbols can also be used instead of string.

ciRunPrecondition

```
ciRunPrecondition(
     g_cache
     t_cGenName
     l_instsNetsPins
     )
     => l_templates / t / nil
```

Description

Runs all the precondition checks that are needed before a constraint generator can be run.

Arguments

g_cache	The constraints cache.
t_cGenName	Name of the constraint generator.
l_instsNetsPins	The list of instances, pins, nets, and terminals that need to be passed to the precondtion. It is of the following format: `list(list("inst1" 'inst) list("inst2" 'inst) list("net1" 'net))`

Value Returned

l_templates	A list of template is returned only if the precondition check for '`createEditReplace` returns `'EditIfExists` and a template that can be edited is found. Currently, the list can contain only one template.
`t`	If the precondition check for '`createEditReplace` returns `'ReplaceIfExists`, the template that needs to be replaced is deleted before returning `t`. If the preconditon fails it will returns nil. Warnings may also be given depending on the precondition.
`nil`	The preconditon failed. Warnings might also be displayed depending on the precondition.

Example

```
ciRunPrecondition(ciGetCellView() "GenericModgen" '('("M1" inst) '("M2" inst)
'("INN" net))) ;; returns t if the precondition passes
```

```
t
```

ciSameCellIterator

```
ciSameCellIterator(
     d_cellView
     t_finderDeviceExpr
     )
     => list
```

Description

Used by the *Circuit Prospector* to iterate over all design instances with the same master collecting them together into groups based on the result of evaluating the passed expression (`finderDeviceExpr`). This iterator ensures that the devices within a group have the same master. All devices with the same master, which have the same `finderDeviceExpr` result, are grouped together. If the `finderDeviceExpr` evaluates to `nil` then the device is ignored.

The `ciSameCellIterator` function evaluates the `finderDeviceExpr` function with the current design instance assigned to a local variable named "device". The device variable can then be referenced in the `finderDeviceExpr`.

Arguments

d_cellView	Cellview containing same cells to be iterated.
t_finderDeviceExpr	Finder device (matched) expression to be used to iterate.

Value Returned

list List of results.

For example:

```
list(
    list( instA1_inst instA2_inst ...) ;;;
    inst group A
    list( instB1_inst instB2_inst ...) ;;;
    inst group B
)
```

Example

To group all fets with the same master according to the net connected to the gate:

```
finderDevExpr = "if(ciIsDevice) then ciNetOnTerm(device\"G\") else nil)"
fetgroups = ciSameCellIterator(cv finderDevExpr)
fetGroup1 = car(fetGroups)
print(fetGroup1~>name)
("fet1" "fet2" "fet3")
```

ciSeparateInstsNetsPins

```
ciSeparateInstsNetsPins(
     l_instsNetsPins
     )
     => l_disembodiedInstsNetsPins
```

Description

Converts the `instsNetsPins` list into a disembodied property list to allow easier access to the instances, nets, pins, and instTerms in the sub-lists. This function is used within constraint generators.

Arguments

l_instsNetsPins	Specifies the list of instances, nets, and pins.

Value Returned

l_disembodiedInstsNetsPins	Returns the disembodied property list.

Example

```
inp = '(("/MPS" inst) ("/MP7" inst) ("/MP" inst) ("/net014" net) ("/VDD" net))
res = ciSeparateInstsNetsPins(inp)


res->insts
("/MPS" "/MP7" "/MP")


res->nets
("/net014" "/VDD")


res->pins
nil


res->instTerms
nil
```

ciSeriesResistorArrayIterator

```
ciSeriesResistorArrayIterator(
     d_cellview
     t_matchExpr
     )
     => l_returnedInsts / nil
```

Description

Iterates over all resistor devices, collating them into groups of sequentially-arranged resistors that create the longest serial chain between two nets. The serial chain splits at a T-junction point. This function is used by the *Series Resistor Array* finder of the Circuit Prospector assistant.

Arguments

d_cellview	The cellview containing the design instances to be iterated over.
t_matchExpr	The matched expression string to be used in the iteration.

Value Returned

l_returnedInsts	A list of returned instances. For example: `list(list( inst1 inst2 inst3 ...))`
`nil`	The device was ignored.

Example

For the *Series Resistor Array* finder,

```
matchExpr = "ciIsDevice(device \"resistor\")"
resDevices = ciSeriesResistorArrayIterator(geGetEditCellView() matchExpr)
print(resDevices~>name)
```

ciSetCMCGSKILLCallbacks

```
ciSetCMCGSKILLCallbacks(
     t_generateConstraintGroupFunctionName
     t_listGeneratableConstraintGroupsFunctionName
     t_constraintGroupCanBeUsedFunctionName
     )
     => t / nil
```

Description

Controls the constraint groups that are visible in the Constraint Manager.

The following types of controls are available:

- Constraint groups that can be viewed in a particular Constraint Manager menu accessible from the Design and Technology databases.
- Constraint groups that can be viewed in a particular Constraint Manager menu, which can be generated.
- Content of the constraint groups that have been generated.

Arguments

t_generateConstraintGroupFunctionName

Name of the function for generating constraint groups. The function should have the signature (`ciCon`, `CGDefName`, `CGName`) and should return a SKILL Constraint Group Object. If the function name is set to `nil`, the default function, `ciGenerateConstraintGroup()`, will be called.

t_listGeneratableConstraintGroupsFunctionName

Name of the function for listing the constraint group names that can be generated. The function should have the signature (`ciCon`, `CGDefName`) and should return a list of strings. If the function name is set to `nil`, the default function, `ciListGeneratableConstraintGroups()`, will be called.

t_constraintGroupCanBeUsedFunctionName

Name of the function for determining whether a given constraint group can be used. The function should have the signature (`ciCon`, `CGDefName`, `CG`) and return `t` or `nil`. If the function name is set to `nil`, the default function, `ciCanCGBeUsed()`, will be called.

Value Returned

`t`	The functions have been registered successfully.
`nil`	Functions have not been registered.

Examples

Example 1

```
ciSetCMCGSKILLCallbacks(nil nil "myCanUse")
```

Use the default callbacks for constraint group generation and listing constraint groups that are generated. Override the default function for determining if a constraint group can be used.

Example 2

```
ciSetCMCGSKILLCallbacks("myGen" "myListGen" "myCanUse")
```

Override all the default callbacks, where `myCanUse`, `myListGen`, and `myGen` can be defined as following:

```
procedure(myCanUse(con cgDefName cg)
    ciCanCGBeUsed(con cgDefName cg) ; Just use the default callback for all cases
)


procedure(myListGen(con, defName)
; we only want to change the shielding list,
; for others we use default that Cadence supplies
    if(strcmp(defName, "shielding") == 0 then
        list("myParallelShield"  "Width Aware Tandem")
    else
        ciListGeneratableConstraintGroups(con, defName)
; Use the default callback for all other cases
    )
)
```

```
procedure(myGen(con, defName, cgName)
    let((cv net cache cg netName)
    ; we only want to change the shielding gen, for others we use
    ; default that Cadence supplies
    if(strcmp(defName, "shielding") == 0 then
        cache = con~>cache
        cv = dbFindOpenCellView(ddGetObj(car(cache~>design)) cadr(cache~>design)
        caddr(cache~>design))
        netName = car(cadr(con~>members))
        net = dbFindNetByName(cv netName)
    if(cgName == "myParallelShield"  then
        cg = myParallelGen(net, cgName)
    else
        cg = myShieldingGen(net, cgName)
        )
        cg
    else
        cg = ciGenerateConstraintGroup(con, defName, cgName)
; Use the default callback for all other cases if cg
        )
    )
)
```

ciSetDefaultConstraintEditor

```
ciSetDefaultConstraintEditor(
     t_ConstraintEditorName
     )
     => t / nil
```

Description

Sets the specified editor as the default constraint editor. The change will take effect at the next start-up of the Virtuoso Schematic Editor or Virtuoso Layout Editor. To keep the change persistent on each start-up of Virtuoso, set this function in an initialization file.

Arguments

t_ConstraintEditorName	Name of the editor that needs to be made the default constraint editor. By default, *Module Generator* is set as the default constraint editor. Following are the possible values for this argument: ■ `Cell Planner...` ■ `Module Generator...` ■ `Process Rule Editor...` ■ `Constraint Comparison Report`

Value Returned

`t`	The value specified for the *t_ConstraintEditorName* argument is a valid constraint editor name and has therefore been successfully set as the default.
`nil`	The specified argument value is an invalid constraint editor name and therefore, the SKILL function fails with `nil` as the return value. In addition, an error message is displayed stating the problem and the probable solution.

Examples

- The *Process Rule Editor* gets set as the default constraint editor successfully because the value for the *t_ConstraintEditorName* argument was specified correctly:

  ```
  ciSetDefaultConstraintEditor("Process Rule Editor...")
      t
  ```

- The name of the default editor was specified incorrectly (note the missing dots at the end of the constraint editor's name) for the *t_ConstraintEditorName* argument. Therefore, the SKILL function returns `nil` and an error message is displayed.

  ```
  ciSetDefaultConstraintEditor("Process Rule Editor")
      nil
      *Error* ciSetDefaultConstraintEditor: (CMGR-3138): Cannot change the
      default constraint editor because the specified constraint editor 'Process
      Rule Editor' does not exist. To change the default constraint editor, call
      function 'ciSetDefaultConstraintEditor' with one of the following
      constraint editor names and then restart Virtuoso Schematic Editor or
      Virtuoso Layout Editor:
          - 'Cell Planner...'
          - 'Module Generator...'
          - 'Process Rule Editor...'
          - 'Constraint Comparison Report'
  ```

ciSetStructArgVal

```
ciSetStructArgVal(
     s_structType
     t_argName
     g_newValue
     )
     => g_success
```

Description

Replaces the specified structure argument value with the specified value. This SKILL function can be used to override the default settings for the predefined structures.

Arguments

s_structType	The structure type symbol, for example, `'CurrentMirror`.
t_argName	The name of the argument for which the value is to be updated.
g_newValue	The new value for the named argument.

Value Returned

g_success	Boolean value indicating success or failure of the argument value update. **Note:** If there is any problem in running the SKILL function, `nil` is returned and warnings are displayed.

Example

```
ciSetStructArgVal('GenericModgen "Horizontal Spacing" 0.001)
ciSetStructArgVal('GenericModgen "Route" t)
ciSetStructArgVal('GenericModgen "Pattern" "ABBA/BAAB")
```

ciSignalIterator

```
ciSignalIterator(
     d_cellView
     t_matchExpression
     )
     => l_signals / nil
```

Description

Used by the Circuit Prospector to iterate over all signals in the passed cellview, and return a list of corresponding signals.

The signals are grouped according to the result of the match expression. If the match expression evaluates to `nil`, the signal list is ignored.

Arguments

d_cellView	Design cellview that contains the signals to be iterated (cellview->signals).
t_matchExpression	A match expression used to group results together, or spread them into separate bins according to the result. **Note:** The local variable “signal”, the value of which is the current signal in the iteration, can be included in the match expression.

Value Returned

l_signals	The list of signal lists grouped as per the evaluated result of the match expression.
`nil`	No results found.

Example

To group all signals from the current schematic cellview according to their signal type:

```
ciSignalIterator(geGetEditCellView() “signal->sigType”)
```

ciUnexpandDeviceInfo

```
ciUnexpandDeviceInfo(
     l_deviceInfo
     [ ?unexpandIterated g_unexpandIterated ]
     )
     => l_unexpandedDeviceInfo
```

Description

Contracts any mfactored device names and optionally any expanded iterated device names in a device information list returned by ciCollectDeviceInfo or ciExpandIteratedDeviceInfo. The mfactored device names used in the layout are of the format `|`*`<instName>.<mFactorIndx>`*, such as `|MN1.3`, and the iterated device names may be schematic device names like `MN1<0>` or layout device names like `|MN1(0)`. In the layout, the device names may be mfactored and iterated, such as `|MN1(3).4`.

When a collection of mfactored device names are contracted (for example, `|MN1.2`, `|MN1.3`, `|MN1.4`) a single device will appear in the returned *`deviceInfo`* list using the base device name like `|MN1`, and the mfactor property will be set to the total number of devices that were contracted, in this case `3`.

Arguments

`l_deviceInfo`	Device information list returned by calling `ciCollectDeviceInfo` or `ciExpandIteratedDeviceInfo`.
`?unexpandIterated` *`g_unexpandIterated`*	Boolean to control whether iterated devices should be contracted or not.

Value Returned

`l_unexpandedDeviceInfo`	A device information list where the mfactored device names and optional iterated device names have been contracted.

Example

Collect the device information for iterated and mFactored layout devices `|MN1(0)`, `|MN1(1)`, `|MN2.3`, `|MN2.4`, and `|MN2.5`:

```
devInfo = ciCollectDeviceInfo(cache '(("|MN1(0)" inst)("|MN1(1)" inst)("|MN2.3"
inst)("|MN2.4" inst)("|MN2.5" inst)))
```

```
mapcar(lambda((dev) list(dev->name dev->mFactor)) devInfo->devs) => '(("|MN1(0)"
1) ("|MN1(1)" 1) ("|MN2.3" 1) ("|MN2.4" 1) ("|MN2.5" 1))
```

Then, compress the device information for iterated and mFactored devices so that the device information relating to the same base device are combined. The combined device name will be modified to reflect the combined iteration range, and the device mFactor will represent the combined mFactor, as shown below:

```
unexpandedDevInfo = ciUnexpandDeviceInfo(devInfo ?unexpandIterated t)
```

```
mapcar(lambda((dev) list(dev->name dev->mFactor)) unexpandedDevInfo->devs) =>
'(("|MN1(0:1)" 1) ("|MN2" 3)) ;;; devs is a list of 2 devices
```

ciUnexpandIteratedDeviceInfo

```
ciUnexpandIteratedDeviceInfo(
     l_deviceInfo
     )
     => l_unexpandedDeviceInfo
```

Description

Contracts any expanded iterated device names in a device information list returned by ciCollectDeviceInfo or ciExpandIteratedDeviceInfo. The iterated device names may be schematic device names, such as `MN1<0>`, or layout device names, such as `|MN1(0)`.

Arguments

l_deviceInfo	Device information list returned by calling `ciCollectDeviceInfo` or `ciExpandIteratedDeviceInfo`.

Value Returned

l_unexpandedDeviceInfo	A device information list where the iterated device names have been contracted.

Example

Collect the device information for individual bits of an iterated device, for example, `MN1<0>`, `MN1<1>`:

```
devInfo = ciCollectDeviceInfo(cache '(("MN1<0>" inst)("MN1<1>" inst) ("MN2"
inst)))
mapcar(lambda((dev) dev->name) devInfo->devs) => '("MN1<0>" "MN1<1>" "MN2")
```

Then, compress the device information for iterated devices so that the device information relating to the same base device are combined. The combined device name will be modified to reflect the combined iteration range, as shown below:

```
unexpandedDevInfo = ciUnexpandIteratedDeviceInfo(devInfo)
mapcar(lambda((dev) dev->name) unexpandedDevInfo->devs) =>
                    '("MN1<0:1>" "MN2") ;;; devs is a list of 2 devices
```

ciUnregisterConstraintGenerator

```
ciUnregisterConstraintGenerator(
     t_constraintGeneratorName
     )
     => t / nil
```

Description

Unregisters a previously registered constraint generator.

If the constraint generator is included in the Constraint Manager's *Constraint Generator* drop-down menu, then using this command will remove the constraint from this list.

See also ciRegisterConstraintGenerator.

Arguments

t_constraintGeneratorName	The name of the constraint generator to be unregistered.

Value Returned

`t`	Constraint generator successfully removed from constraint creation pull-down in the *Constraint Manager*.
`nil`	Action failed. Constraint generator was not removed.

Example

```
ciUnregisterConstraintGenerator("Matching (strength)")
```

Removes the constraint generator entitled *Matching (strength)*.

ciUnregisterIterator

```
ciUnregisterIterator(
     t_iteratorFnName
     )
     => t / nil
```

Description

Unregisters a Circuit Prospector iterator that was previously registered by using ciRegisterIterator. If successful, the iterator will no longer appear in the *Edit Finder* list or the *Edit Iterator* list.

Arguments

t_iteratorFnName	The name of the iterator function to use.

Values Returned

`t`	Iterator successfully unregistered.
`nil`	Iterator not unregistered.

Example

```
ciUnregisterIterator("Pin Iterator")
t
```

ciVariantInfoForFingersAndFingerWidth

```
ciVariantInfoForFingersAndFingerWidth(
     u_cache
     l_instsNetsPins
     [ ?minFingerWidth t_minWidth ]
     [ ?maxFingerWidth t_maxWidth ]
     )
     => nil
```

Description

Used by the Circuit Prospector as a generator to set a variantInfo property to each given instance registered as a fet, when the parameter value for the parameter registered as fingerWidth is greater than the given minWidth value, and smaller than the maxWidth value.

- The value of the *variantInfo* property is a disembodied list with a pair of parameters “params” and “mode”.
- The value of the parameter “params” is a pair list of parameters and values.
- The first parameter of that latter pair list is the name of the parameter found on the instance and registered as the “fingerCount” parameter. Its value is a list of two or three variant values for the parameter that corresponds to the number of fingers. One of the values is the original value. The other values are half and double the original value, when the result is even.
- The second parameter of that pair list is the name of the parameter found on the instance and registered as the “fingerWidth” parameter. Its value is a list of two or three variant values for the parameter that corresponds to the finger width. One of the values is the original value. The other values are determined by the variant values for the number of fingers to keep the same total width value.
- The value of the parameter “mode” is “paramSets”. When that property is set, the Analog Placer automatically picks any variant for instantiation.

Arguments

u_cache	Constraint cache where the instances can be found from their full path names (See `cache`.).
l_instsNetsPins	List of instance members with their full path names and type, as selected in the *Circuit Prospector* browser.

`?minFingerWidth` *`t_minWidth`*	Optional value for the minimum value of the finger width. Default: `"0.5u"`
`?maxFingerWidth` *`t_maxWidth`*	Optional value for the maximum value of the finger width. Default: `"30u"`

Value Returned

`nil`	Always `nil`.

Example

If `MN1`, `MN6`, and `MP4` are the names of instances registered as `fet`. Additionally, the parameter `nf` is registered as `fingerCount`, and the parameter `fw` is registered as `fingerWidth`, then when the value of `nf` is “8” and the value of `fw` is “10u”, the following functions create a *variantInfo* property for `MN1`, `MN6`, and `MP4`, and the value of that property will be `(params (("nf" (4 8 16)) ("fw" (2e-05 1e-05 5e-06))) mode "paramSets")`:

```
cache = ciCacheGet(geGetEditCellView()
ciVariantInfoForFingersAndFingerWidth(cache list( list(“/MN1” `inst) list(“/MN6”
`inst) list(“/MP4” `inst)) ?minFingerWidth “4u” ?maxFingerWidth “60u”)
```

ciXYInstSymmetricIterator

```
ciXYInstSymmetricIterator(
     d_cellview
     t_finderExpr
     [ ?trigger g_trigger ]
     [ ?likeSchemaSym g_likeSchemaSym ]
     )
     => list / nil
```

Description

Evaluates the `finderExpr` with the current symmetric pair of objects that are assigned to `L` and `R` local variables. Used by *Circuit Prospector* finders to iterate over all symmetric design instance pairs, collecting them into symmetric pairs if the result of evaluating the passed expression (`finderExpr`) is not `nil`.

The `L` and `R` variables can be referenced in the `finderExpr`.

In the current schematic, the instance symmetry iterator first looks for symmetrical triggering pairs and then propagates the symmetries from these pairs along the nets and devices that are connected to the triggering pairs and symmetrical pairs which can be one of the following:

- A differential pair made of `fet` and `bjt` devices.
- A pair of instances with the same cell name and same size as `fet` or `bjt` devices with mirrored orientation, aligned on the same Y co-ordinate.

The symmetries are propagated through the nets using terminal names defined for each active device. Symmetries are also transmitted to and propagated through passive devices.

To be a symmetrical pair, both instances must have the same cell name and size.

Symmetries for instances are converted to self-symmetries when there is only one member on the path for symmetry propagation.

Arguments

`d_cellview` — The cellview that *Circuit Prospector* finders are to iterate over all design instance pairs.

`t_finderExpr` — The finder expression to be used.

`?trigger g_trigger` — A trigger to capture the symmetries. If set to `t`, the default, the found differential pairs will be used to trigger the capture of the symmetries.

If set to `nil`, the differential pairs are not used to trigger the capture of symmetries. Only the pairs of active devices with mirrored orientation, aligned on the same Y coordinate, and without connection to a power supply, will be used to trigger the capture of symmetries.

`?likeSchemaSym g_likeSchemaSym`

Sets the orientation of the symmetries triggered by the mirrored active devices.

When set to `t`, the default, the symmetries order is the same as on the schematic. That is, the first member of each matching symmetrical pair should be on the left of the symmetrical axis.

When set to `nil`, the order of the symmetry members is reversed.

Value Returned

`list` — List of symmetric pairs, for example:

```
list(
    list( objA1 objA2 ;;; symmetric pair A
    list( objB1 objB2 ;;; symmetric pair B
    ...
)
```

`nil` — Command failed.

Example

```
finderExpr="(L->libName == R->libName) && (L->cellName == R->cellName) &&
(L->w == R->w) && (L->l == R->l) && (L->r == R->r) && (L->c == R->c)"

symmPairs  = ciXYInstSymmetricIterator(cv finderExpr)

symmPair1  = car(symmPairs)
print(symmPair1~>name)
("MN16" "MN5")
```

ciXYNetSymmetricIterator

```
ciXYNetSymmetricIterator(
     d_cellview
     t_finderExpr
     [ ?trigger g_trigger ]
     [ ?likeSchemaSym g_likeSchemaSym ]
     )
     => list / nil
```

Description

Used by *Circuit Prospector* finders to iterate over all pairs of symmetric nets, collecting them into symmetric pairs if the result of evaluating the passed expression (`finderExpr`) is not `nil`.

In the current schematic, the net symmetry iterator will first of all look for symmetrical triggering pairs and propagate the symmetries from these pairs along the nets and devices that are connected to the triggering pairs and the following symmetrical pairs.

The symmetries are propagated through the nets using terminal names defined for each active device.

Symmetries for nets are converted to self-symmetries when there is only one member on the path for symmetry propagation.

Arguments

d_cellview	The cellview that *Circuit Prospector* finders are to iterate over all symmetric nets.
t_finderExpr	The finder expression to be used.
`?trigger` *g_trigger*	If set to `t`, the default, the found differential pairs will be used to trigger the capture of the symmetries. If set to `nil`, the differential pairs will be ignored to trigger the capture of symmetries. Only the pairs of active devices with mirrored orientation, aligned on the same Y coordinate and without connection to a power supply, will be used to trigger the capture of symmetries.

`?likeSchemaSym` *g_likeSchemaSym*

Boolean argument that refers to the preferred order in which to store symmetry. If set to `t`, then symmetry is one-way, otherwise symmetry will be both ways.

Value Returned

list	List of symmetric pairs, for example:
	`list(`
	`    list( netA1 netA2 ;;; symmetric pair A`
	`    list( netB1 netB2 ;;; symmetric pair B`
	`    ...`
	`)`

`nil`	Command failed.

Example

```
finderExpr="t"
symmPairs  = ciXYNetSymmetricIterator(cv finderExpr)

symmPair1  = car(symmPairs)
print(symmPair1~>name)
("net1" "net2")
```

ciXYPinSymmetricIterator

```
ciXYPinSymmetricIterator(
     d_cellview
     t_finderExpr
     [ ?trigger g_trigger ]
     [ ?likeSchemaSym t_likeSchemaSym ]
     )
     => list / nil
```

Description

Used by *Circuit Prospector* finders to iterate over all pairs of symmetric pins, collecting them into symmetric pairs if the result of evaluating the passed expression (`finderExpr`) is not `nil`.

In the current schematic, the pin symmetry iterator will first of all look for symmetrical triggering pairs and propagate the symmetries from these pairs along the nets and devices that are connected to the triggering pairs and the following symmetrical pairs.

Symmetries for pins are converted to self-symmetries when there is only one member on the path for symmetry propagation.

Arguments

d_cellview — The cellview that *Circuit Prospector* finders are to iterate over all symmetric nets.

t_finderExpr — The finder expression to be used.

?trigger *g_trigger* — If set to `t`, the default, the found differential pairs will be used to trigger the capture of the symmetries.

If set to `nil`, the differential pairs will be ignored to trigger the capture of symmetries. Only the pairs of active devices with mirrored orientation, aligned on the same Y coordinate and without connection to a power supply, will be used to trigger the capture of symmetries.

?likeSchemaSym *t_likeSchemaSym*

Boolean argument that refers to the preferred order in which to store symmetry. If set to `t`, then symmetry is one-way, otherwise symmetry will be both ways.

Value Returned

`list` List of symmetric pairs, for example:

```
list(
    list( objA1 objA2 ;;; symmetric pair A
    list( objB1 objB2 ;;; symmetric pair B
    ...
)
```

`nil` Command failed.

Example

```
finderExpr="t"
symmPairs  = ciXYPinSymmetricIterator(cv finderExpr)

symmPair1  = car(symmPairs)
print(symmPair1~>name)
("I153" "I154")
```

ciXYSortInsts

```
ciXYSortInsts(
     l_list(dbInstId)
     )
     => l_list(dbInstId) / nil
```

Description

Sorts a list of `dbInstID`, first by their X coordinates and then by their Y coordinates. This will order the instances, from top to bottom and left to right, in terms of where they are located on the canvas.

Arguments

l_list(*dbInstId*)	The instance database identfier.

Value Returned

l_list(*dbInstId*)	List of `dbInstID` correctly sorted.
`nil`	Command failed.

Example

```
insts~>xy
 ((-3.5625 1.125)
  (-2.4375 0.875)
  (-1.4375 1.125)
  (-3.125 1.5625)
  (-1.8125 1.5625)
  (-2.9375 2.0625)
  (-2.9375 2.5625))

xySortedInsts = ciXYSortInsts(insts)

xySortedInsts~>xy
 ((-2.9375 2.5625)
  (-2.9375 2.0625)
  (-3.125 1.5625)
  (-1.8125 1.5625)
```

```
(-3.5625 1.125)
(-1.4375 1.125)
(-2.4375 0.875))
```

ciXYSymmetricIterator

```
ciXYSymmetricIterator(
     d_cellView
     t_finderExpr
     )
     => list / nil
```

Description

Evaluates the `finderExpr` with the current symmetric pair of objects assigned to `L` and `R` local variables. The `L` and `R` variables can be referenced in the `finderExpr`. Iterates all pairs of symmetric design instances/pins (objects) collecting them together into symmetric pairs if the result of evaluating the passed expression (`finderExpr`) is not `nil`. The *Circuit Prospector* assistant finders use this function.

Arguments

d_cellView	Cellview containing pairs of symmetric design objects to be iterated.
t_finderExpr	Finder (matched) expression to be used to iterate.

Value Returned

list — List of results. For example:

```
list(
    list( objA1 objA2) ;;; symmetric pair A
    list( objB1 objB2) ;;; symmetric pair B
    ...
)
```

`nil` — Failed to generate list.

Example

To find all symmetrical devices which share the same library/cell name:

```
finderExpr = “L->libName == R->libName && L->cellName == R->cellName”
symmPairs = ciXYSymmetricIterator(cv finderExpr)
symmPair1 = car(symmPairs)
print(symmPair1~>name)
(“inst1” “inst2”)
```

Rapid Analog Prototype Category (Circuit Prospector) Customization SKILL Commands

Customizing the RAP Finders/ Generators

- Each of the modgen based finder/generators in the RAP category call the `ciCreateModgen()` function, which is the high level entry point for modgen generation.
- The `ciCreateModgen()` function evaluates the modgen pattern expression that has been pre-registered for the specified structure type as well as handling device abutment, what type of guardring to add and whether to add dummies or not. It is possible to register alternative modgen pattern expression using `ciSetStructGeneratorExpressions()`.
- The args list supplied to `ciRegisterConstraintGenerator()` supports separators to make the popup constraint generator options dialog easier to read. This is achieved by using the `'separator` argument type. For example:

```
args= ( …
    list("Guard Ring Options" 'separator)
…)
```

- The args list supports expandable and collapsable groups of arguments. This is achieved by using the `'beginExpandedOptions` and `'endExpandedOptions` argument types. For example:

```
args= ( …
    list("Guard Ring Options" 'beginExpandedOptions)
    … guard ring args specified between begin/end …
    list("Guard Ring Options" 'endExpandedOptions)
…)
```

- An optional `'hide` parameter is supported on each argument to control the visibility of the argument on the dialog. This allows you to trim down and simplify the dialogs where default values of arguments are deemed sufficient and/or the user should be prevented from altering these values. For example:

```
args= ( …
    list("Guard Ring Type" 'enumlist("none" "ring" "pane" "ring") 'hide)
)
```

- Multi-line text boxes can be created using the following argument types:
  - `'multiString`
  - `'pattern`

- ❑ `'orient`

The text box created using the `'multiString` argument type allows use of variable-sized font; whereas, the one created using the `'pattern` or `'orient` argument type use fixed-size font only.

The `'pattern` and `'orient` arguments types are useful for entering interdigitation patterns and device orientations. This is because the fixed-size font makes it easier to align device symbols and orientations on multiple lines.

For example,

```
list("Description" ‘ "\" A description\n On multiple lines\"")
list("Device Interdigitation” ‘pattern "\"ABBA/BAAB\"")
     ;;; Use / or \n for new line
list("Device Orientation” ‘orient "\"R0 R0 R0 R0/MX MX MX MX\"")
     ;;; Use / or \n for new line
```

- You can specify `'wigdetType` as `comboBox` for a `'string` argument type as shown below:

```
list(argName 'string "buildString(Expression)" 'widgetType "comboBox")
```

As a result, a drop-down list box gets added to the generator GUI that contains values based on the specified `Expression`. At run time, when you open the generator GUI, the specified `Expression` is evaluated and the values are displayed dynamically.

Customizing the RAP Finders: New Functions

A set of functions exist to allow the arguments lists to be fully customizable and PDK independent:

- `ciGetStructTypes()`: Returns a list of registered structure types. For example:

```
list('DiffPair, 'CurrentMirror, 'CascodedCurrentMirror, 'LargeMfactor,
'PassiveDeviceArrray)
```

- `ciAddStructArg(<structSymbolName> <argDef> @key (afterArgName nil))`: Adds the argument definition to the end of args or after the specified `argName`.
- `ciGetStructArg(<structSymbolName> <argName>)`: Gets a specified argument definition.
- `ciReplaceStructArg(<structSymbolName> <argName> <argDef>)`: Replaces a specified argument definition.
- `ciDeleteStructArg(<structSymbolName> <argName>)`: Deletes a specified argument definition.

- `ciGetRoutingLayers()`: Retrieves the list of routing layer names from the technology file. For example:

  ```
  list("Metal1" "Metal2" "Metal3" "Metal4" "Metal5" "Metal6" "Poly")
  ```

- `ciCreateRoutingLayerEnumString(layerNumber)`: Is a utility function for creating layer name enums, which retrieves the list of layer names from the technology file and make layerNumber the default by adding the associated layer name to the end of the list. For example:

  ```
  ciCreateRoutingLayerEnumString(2) => list("Metal1" "Metal2" "Metal3" "Metal4"
  "Metal5" "Metal6" "Poly" "Metal2")
  ```

- `ciGetRule(layerName ruleName defaultValue)`: Retrieves the required rule value from the technology file for a given layerName, where:
  - `layerName`: The layer name to get the rule value for.
  - `ruleName`: The ruleName.
  - `defVal`: A default value in case the rule is not found. If a default is not specified, then an error will be reported if the rule is not found.

For example:

```
ciGetRule("Metal2" "minWidth" 0.123) => 0.3 (techfile value)
ciGetRule("Metal2" "minWidth" 0.123) => 0.123 (default value)
ciGetRule("Metal2" "minWidth" 0.123) => 0.123 (default value)
```

Customizing the RAP Finders for Modgens: MOS Current Mirror

To simplify the definition of structures that require dummies and abutment, sets of arguments are pre-registered for each of these using the struct types `'Dummies`, `'Abutment`, and `'GuardRing`. For example:

```
ciSetStructArgs('CurrentMirror
    append(
    append(
    append(
    append(
    list(list("Style" `enum "\"Auto SingleRow DoubleRow\""))
ciGetStructArgs('Dummies)) ciGetStructArgs('Abutment))
ciGetStructArgs('GuardRing))
)
)
```

- where, `ciGetStructArgs('Dummies) -> list( list("Add Dummies" 'bool nil))`
- where, `ciGetStructArgs('Abutment) -> list( list("Abut All" 'bool t))`

- where, `ciGetStructArgs('GuardRing) ->`

```
list(
    list(“Guard Ring" 'separator) ;;; Adds a separator into the GUI --Guard
    Ring ----------
    list("Add Guard Ring” 'bool nil)
    list(“Settings" 'beginExpandedOptionsnil) ;;; Adds an expander >>> into the
    GUI
    list(“Type” 'enum “\“ring pane\””)
    list(“Shape” 'enum “\”rectangular rectilinear\””)
    list(“Net” 'enum “buildString(ciGetMembersOfType(instsNetsPins'net
    ?includeTypenil))”)
    list(“Spacing” `float “0.0”)
    list(“MPP”
    `enum "strcat(buildString(techGetMPPTemplateNames(ciGetTechFile())) \" \"
    ciGetGuardRingMPPName(car(ciInstsNetsPinsToDevInfo(cache instsNetsPins)-
    >devs)->dbId))")
    list("Fluid Guard Ring" `separator) ;;; Adds a separator into the GUI --
    Fluid Guard Ring ----------
    list(“Use Fluid" `bool nil)
    list(“Device” 'enum "buildString(ciGetMembersOfType(instsNetsPins 'inst
    ?includeType nil))")
    list(“Width” `float 0.0)
    list(“Use Min DRC For Spacing” 'bool nil)
    list(“Settings" 'endExpandedOptions nil)
)
```

- Where, `ciGetStructPDKMult(structType)` returns a PDK dependent multiplier, which can be applied to the values returned by `ciGetRule()`. This function returns `1.0` when called with `structType` set to `'Default`.

- Different multipliers can be set for different structTypes and PDKs using `ciSetStructPDKMult(structType pdkName multiplier)`. For example: `ciSetStructPDKMult('CurrentMirror" gpdk045" 1.0)`

  Note: The `ciGetStructPDKMult()` function dynamically determines the PDK name when the expression is evaluated.

- The `ciCreateModgen()` function calls expressions to create the modgen device pattern, and guard ring. These expressions are pre-registered for each of the structure types.

- It is possible to register alternative modgen pattern, and guard ring expressions using `ciSetStructGeneratorExpressions()`.

- A SKILL structure `ciStructGeneratorExpressions` can be defined with pattern, routing, and guardRing fields. These fields are SKILL expressions that are called to generate the modgen device pattern, and guardRing.

- By default the MOS Current Mirror generator expressions are registered as follows:

```
ciSetStructGeneratorExpressions('CurrentMirror
    make_ciStructGeneratorExpressions(
    ?pattern "ciGenerateCurrentMirrorPattern(devInfo args)"
    ?guardRing"when(args->\"Add GuardRing\" ciCreateGuardRing(cache
    modgen args))"
    )
)
```

Where,

- `ciGenerateCurrentMirrorPattern()` returns either single row or double row device pattern lists dependent on the settings in the generator args.
- `ciCreateGuardRing()` creates a guard ring for the modgen based on the guard ring settings in the passed generator args.

Customizing the RAP Finders for Modgens: ciStructGeneratorExpressions-> pattern

The pattern field expression of the `ciStructGeneratorExpressions` structure should evaluate to a lists of lists where each sub-lists. Each sublist represents a row in the modgen and contains device names and optionally a list of modgen device parameters. The format of the device names and parameters is similar to that used when specifying the members when calling `ciConCreate()`, although the member type is not required.

The registered pattern expression can reference the cache, `devInfo`, and args as variables. Where, `devInfo` is the device information DPL returned by the `ciCollectDeviceInfo()` function and `args` is a disembodied property list containing the constraint generator `args` and values specified when the constraint generator was run. For example, to represent the pattern `ABBA`/`BAAB` for a diff pair with mfactor4, with devices named “`MN1`” and “`MN2`”, and custom spacings for some of the devices in the pattern the pattern list would be:

```
'( (("MN1" (("horizontalCustomSpacing" 0.2))) ("MN2" (("horizontalCustomSpacing"
0.3))) ("MN2") ("MN1"))
(("MN2" (("horizontalCustomSpacing" 0.2))) ("MN1" (("horizontalCustomSpacing"
0.3))) ("MN1") ("MN2")) )
```

The `ciCreateModgen()` function takes this information and assign the correct row/column numbers and modgen pattern string for the modgen being created.

The `devs` field of the `devInfo` variable is a list of disembodied property lists where each disembodied property list contains PDK independent information about a device in the structure. This information can be used to generate the required modgen pattern lists.

For example, to generate a single row modgen containing all devices in the structure you could write a procedure as follows:

```
procedure( createSingleRowPattern(cache devInfo args)
    let( (pattern row)
        row = list()
        foreach(dev devInfo->devs
            for( m1 dev->mFactor
                row = cons(dev->name row )
            )
        )
        pattern = list(reverse(row))
    )
)
```

Then set the pattern expression field of the `ciStructGeneratorExpressions` structure to call this function. For example:

```
genExprs= ciGetStructGeneratorExpressions('CurrentMirror)
genExprs->pattern = “createSingleRowPattern(cache devInfoargs)”
ciSetStructGeneratorExpressions('CurrentMirror genExprs)
```

Another example demonstrating the use of the `devInfo` disembodied property list:

```
;;; ABBA
;;; BAAB
;;;
;;; Note args contains user defined properties spacing1and spacing2 for specifying
the device spacings
;;;
procedure( ciDiffPairPatternMFactor4( cache devInfo args)
    prog( (devAdevBrow0 row1 pattern)
    unless(length(devInfo->devs) == 2 warn(“Wrong number of devices\n”) return())
    devA= car(devInfo->devs)
    devB= car(devInfo->devs)
    unless(devA->mFactor == 4 && devB->mFactor == 4 warn(“wrong mfactor”) return())
    row0 = list( list(devA->name args->spacing1) list(devB->name args->spacing2)
    list(devB->name) list(devA->name))
    row1 = list( list(devB->name args->spacing1) list(devA->name args->spacing2)
    list(devA->name) list(devB->name))
    pattern = list(row0 row1)
    return(pattern)
    )
)
```

Customizing the RAP Finders for Modgens: ciStructGeneratorExpressions -> guardRing

The guardRing field of the `ciStructGeneratorExpressions` structure should be an expression that generates a guard ring for the passed modgen based on the guard ring settings in the passed generator args.

By default the guardRing expression for all structure types are registered as:

```
"when(args->\"Add GuardRing\" ciCreateGuardRing(cache modgen args))"
```

Here is an example of how to register alternative guard ring generator expressions for structure types:

```
procedure( createGuardRing(cache modgen args)
    let( (guardRingNet guardRingMembers guardRingParams guardRing)
        when(args->"Add GuardRing"
            guardRingNet= list(args->Net 'net list( list("shape" args->Shape)
            list("spacing" args->Spacing) list("mppName" args->MPP)))
            guardRingMembers= list(list(modgen->name 'modgen) guardRingNet)
            guardRingParams= list( list( "type" args->Type ) )
            guardRing = ciConCreate(cache 'powerStructure ?members
            guardRingMembers ?params guardRingParams ?verbose nil)
            )
    )
)
genExprs= ciGetStructGeneratorExpressions(‘CurrentMirror)
genExprs->guardRing= "createGuardRing(cache modgen args)"
ciSetStructGeneratorExpressions(‘CurrentMirror genExprs)
```

Examples for Registering a New Structure Type and Generator to Extend the Built-in RAP Finders

This example shows how to register a structure type, such as Cascode, using the customization functions described in the previous section.

- Register the args for this structure type:

```
ciSetStructArgs('Cascode
    append(
    append(
    append(
    append(
list(list("Style" `enum "\"Auto SingleRow DoubleRow\""))
ciGetStructArgs('Dummies)) ciGetStructArgs('Abutment))
ciGetStructArgs('GuardRing)) "ciGetStructPDKMult('Default)"
"ciGetStructPDKMult('Cascode)")))
```

- Register the generator expressions for this structure type:

```
ciSetStructGeneratorExpressions('Cascode make_ciStructGeneratorExpressions(
    ?pattern “generateCascodeModgenPattern(devInfo args)"
    ?guardRing “when(args->\”Add GuardRing\” createCascodeGuardRing(cache
    modgen args)) ))
```

- Register a constraint generator for the new structure:

```
ciRegisterConstraintGenerator(
    list(nil
        'name"Module (Cascode)"
        'description "Generate Modgen Constraint for a Cascode"
        'expression "ciCreateModgen(cache instsNetsPins 'Cascode args)"
        'addToToolbar t
        'iconName"Cascode"
        'args"ciGetStructArgs('Cascode)”
        'menu "Rapid Analog Prototype"
    )
```

Constraint Generator Customization SKILL Commands

The following SKILL commands allow you to customize the available constraint generators:

Commands for the constraint generator structure arguments	
ciAddStructArg	ciRegexReplaceStructArgs
ciDeleteStructArg	ciReplaceStructArg
ciGetStructArg	ciSaveConstraintGenerator
ciGetStructArgs	ciSetStructArgs
Commands for the constraint generator structure argument expressions	
ciCreateRoutingLayerEnumString	ciListTemplateTypes
ciGetGuardRingMPPName	ciNumDevices
ciGetRoutingLayer	ciReinitStructTemplateDefs
ciGetRoutingLayers	ciRemoveSymmetricPinAlignments
ciGetRule	ciSetStructPDKMult
ciGetStructPDKMult	ciUtilsMakeNumberRange
Commands for the constraint generator expressions associated with each type of structure	
ciGetStructGeneratorExpressions	ciSetStructGeneratorExpressions
ciListStructGeneratorExpressions	
Commands for the modgen pattern and guard ring expressions	
ciCollectDeviceInfo	ciGenerateDiffPairPattern
ciConvertParamsDPLToParams	ciGenerateLargeMfactorPattern
ciConvertParamsToDPL	ciGetGuardRing
ciConvertToConArg	ciGetParamValFromParameters
ciCreateGuardRing	ciGUIArgsToConArgs
ciCreateModgen	ciHighestLevelNet
ciCreateModgenDummy	ciListStructPDKMults
ciCreateModgenForStructure	ciListStructTypes
ciDeleteClusterMembersWithinModgens	ciMemberIndexToModgenPatternSymbol

ciDeleteGuardRing	ciModgenDummyNetName
ciDeleteSymmetriesWithinModgens	ciPadModgenPattern
ciExpandAndRepeatName	ciSortDeviceInfoByFingerWidth
ciExtractRowNumber	ciSortDeviceInfoByMfactor
ciFindDeviceArraysForDev	ciSortDeviceInfoByX
ciGenerateArrayChannelDesc	ciSortDeviceInfoByXY
ciGenerateBestFitPattern	ciSortDeviceInfoByY
ciGenerateCascodedCurrentMirrorChannelDesc	ciSortDeviceInfoByYX
ciGenerateCascodedCurrentMirrorPattern	ciUnexpandPhysicalDeviceInfo
ciGenerateCurrentMirrorChannelDesc	ciUpdateModgenParamsAndMembers
ciGenerateCurrentMirrorPattern	ciUtilsGetArgVal
ciGenerateDiffPairChannelDesc	

ciAddStructArg

```
ciAddStructArg(
     s_structType
     l_newArg
     [ ?afterArgName g_afterArgName ]
     )
     => t / nil
```

Description

Adds a new constraint generator argument to the list of existing arguments registered for the passed `structType`. By default, the argument is appended to the current list of arguments for the `structType`. Optionally, the new argument can be inserted after the end of the argument or after a existing argument name.

Arguments

s_structType	The structure type symbol.
l_newArg	A list defining the new argument to be added in the form list (<t_argName> <s_argType> <g_argVal | t_argExpr> [s_'hide])
`?afterArgName` *g_afterArgName*	Optionally insert the new argument after the existing named argument.

Value Returned

`t`	Successfully added a new constraint generator argument to the list of existing arguments registered for the passed `structType`.
`nil`	Operation failed.

Example 1

```
ciAddStructArg('DiffPair '("newArg" 'double 0.0))
```

As per the above example a new argument is appended to the list of arguments for a `DiffPair` structure.

Example 2

```
ciAddStructArg('CurrentMirror '("newArg" 'string "aStringExpression")
?afterArgName "Guard Ring")
```

As per the above example a new argument is inserted in the list of arguments for a `CurrentMirror` structure after the existing argument named, `Guard Ring`.

ciCollectDeviceInfo

```
ciCollectDeviceInfo(
     g_cache
     l_devices
     [ ?devParamNames l_devParamNames ]
     [ ?devTerminalNames l_devTerminalNames ]
     [ ?warnParamUnfound g_warnParamUnfound ]
     [ ?warnUnmapped g_warnUnmapped ]
     [ ?parents g_parents ]
     )
     => devs
```

Description

Returns a disembodied property list containing information collected about the passed devices in the generators instsNetsPins list.

For example,

```
info = ciCollectDeviceInfo(cache '(("M0" inst)("M1" inst)("agnd_h" net)))
```

The information is collected and returned in a PDK independent way so that all device information can be accessed in the same way regardless of PDK. For better performance, the collected information is cached. As a result, any warnings generated during the first call of `ciCollectDeviceInfo` are not displayed again as long as the function is called with the same arguments and the cached information is considered up to date.

This function has optional keyed arguments to specify the device parameters and device terminal connections for which information needs to be collected.

The `ciCollectDeviceInfo` function differentiates between the absolute and the relative path for the specified device. Dots (`..`) cannot be used in the object path to navigate in the hierarchy.

Arguments

g_cache	The constraints cache.
l_devices	A list of the lists where each sublist contains the device name and type for which information is to be collected.

`?devParamNames` *l_devParamNames*

A list of PDK independent device parameter names for which the values should be returned. By default, this list is set to `'(("mFactor" 1) ("fingerCount" 1) ("length" nil) ("width" nil) ("fingerWidth" nil))` where the PDK independent parameter names have been registered with the `ciMapParam()` function.

`?devTerminalNames` *`l_devTerminalNames`*

A list of PDK independent device terminal names for which connectivity information should be returned. By default, this list is set to `'("source" "gate" "drain" "bulk")` where the PDK independent terminal names have been registered with the `ciMapTerm` function.

`?warnParamUnfound` *`g_warnParamUnfound`*

Defaults to `t` and controls whether warnings are issued if a named parameter is not found on a device.

`?warnUnmapped` *`g_warnUnmapped`* Defaults to `t` and controls whether warnings are issued if a named parameter has not been registered with ciMapParam.

`?parents` *`g_parents`* Defaults to `nil`. When set to `t`, this parameter will evaluate `iPar` and `pPar` expressions used in parameter values.

Value Returned

```
Info = ciCollectDeviceInfo(... ...)
```

Here, `Info` is a list of the following format:

```
list(nil
    'devParamNames ( ("paramName1" valueIfNotFound1) ... ("paramNameN"
    valueIfNotFoundN))
    'devTerminalNames ("net1" ... "netN")
    'cache ci:0x26425c20
    'uniqueNets ("uniqueNet1" ... "uniqueNetN")
    'devs ( dev1 ... devN)
    'net1Nets ("net1Name1" ... "net1NameN")
    'net1NetsUnique ("uniqueNet1Name1" ... "uniqueNet1NameN")
```

```
    ...
    'netNNets ("netNName1" ... "netNNameN")
    'netNNetsUnique ("uniqueNetNName1" ... "uniqueNetNNameN")
)
```

The default format of `info->devs` is a list of the following format:

```
'(dev1 ... devX ... devN)
```

Where `devX` is a DPL of the following format:

```
list(nil
    'termNets ("termNetName1" ... "termNetNameN")
    'name <name>
    'dbId  <dbId>
    'deviceTypes  <("deviceTypeName1" ... "deviceTypeNameN")>  | < ('default)>
    'paramName1 <paramValue1>
    ...
    'paramNameN <paramValueN>
    'net1 <nethelem(1 dbId->instTerms)->net->name>
    ...
    'netN <nethelem(N dbId->instTerms)->net->name>
    'termNet1 <"termNet1Name1">
    ...
    'termNetN <"termNet1NameN">
)
```

If a `paramValueN` is not found on the device, the default value specified for that parameter in the `devParamNames` list is used.

- `info->sourceNets`: A list of the source net names attached to the devices.
- `info->gateNets`: A list of the gate net names attached to the devices.
- `info->drainNets`: A list of the drain net names attached to the devices.
- `info->bulkNets`: A list of the bulk net names attached to the devices.
- `info->sourceNetsUnique`: A list of the unique source net names attached to the devices.
- `info->gateNetsUnique`: A list of the unique gate net names attached to the devices.
- `info->drainNetsUnique`: A list of the unique drain net names attached to the devices.
- `info->bulkNetsUnique`: A list of the unique bulk net names attached to the devices.

- `info->uniqueNets`: A list of all the unique net names attached to the devices.

`ciCollectDeviceInfo()` has the following two optional keyed arguments that allow you to specify which device parameters and device terminal connections to collect.

- The `devParamNames` parameter specifies the list of PDK independent device parameter names to be collected on each device. The PDK independent device parameter names are those that have been registered with `ciMapParam()`. For example, `"mFactor"` and `"fingerCount"`.
- Similarly, the `devTerminalNames` parameter specifies the list of PDK independent device terminal names for which connectivity (nets) are to be collected. The PDK independent device terminal names are those that have been registered with `ciMapTerm()`. For example, `"source"` and `"drain"`.

The DPL property names are added dynamically to match whatever device parameter or device terminal names have been specified. Therefore, if the `devParamNames` includes a parameter named `YYY`, the returned device information will have a `YYY` field, that is, `car(info->devs)->YYY=> "value of YYY"`.

How ciCollectDeviceInfo uses the registered information?

The `ciCollectDeviceInfo` function uses any information registered with the following functions: ciDeviceInfoRegisterParams, ciDeviceInfoRegisterTerminals, ciMapParam, ciMapTerm, ciRegisterDevice, and so on.

If no parameters or terminals have been registered with the `ciDeviceInfoRegisterParams` or `ciDeviceInfoRegisterTerminals` function, the `ciCollectDeviceInfo` function works the same as described in the above section. This is the default behavior that has been provided for backward compatibility.

However, if the keyed argument `devParamNames` or `devTerminalNames` has been passed to `ciCollectDeviceInfo`, it overrides any registered parameters or terminals.

If you have registered parameter names or terminals for a device type with `ciDeviceInfoRegisterParams` or `ciDeviceInfoRegisterTerminals`, the following happens:

1. `ciCollectDeviceInfo` gets all device types associated with the device cellview found for the given instance name. For example:

   ```
   ciRegisterDevice("passive"  '(("myLib" "res" nil)))
   ciRegisterDevice("resistor" '(("myLib" "res" nil)))
   ```

   Let us consider `"R0"` is an instance that belongs to the library `"myLib"` and cell name `"res"`. In that case, the function is passed as following:

```
ciCollectDeviceInfo(ciGetCellView() '(("R0" inst)))
```

Here, `ciCollectDeviceInfo` looks for all device types of `"R0"` which are `'(passive resistor).`

2. `ciCollectDeviceInfo` gets all parameters or terminals information for all types of the instance. For example, for `"R0"`, the `ciCollectDeviceInfo` function gets all parameters or terminals information for the `"passive"` and `"resistor"` types in the `ciDeviceInfoRegistry` function.

   a. If nothing has been registered for all device types of the instance, the `ciCollectDeviceInfo` function gathers the information for the parameters or terminals returned by the `ciDeviceInfoGetRegisteredParams('default)` or `ciDeviceInfoGetRegisteredTerminals('default)` function, respectively.

   Consider that Cadence-provided default parameters or terminals have not been overridden. That is,

   - `'(("mFactor" 1) ("fingerCount" 1) ("length" nil) ("width" nil) ("fingerWidth" nil))` for parameters
   - `'("source" "gate" "drain" "bulk")` for terminals

   As a result, if these parameters are not mapped (using `ciMapParam`) to a parameter of the instance, a warning is displayed.

   b. If parameters have been registered for at least one device type of the instance device type, the `ciCollectDeviceInfo` function gathers information for the parameters found in the table returned by the `ciDeviceInfoRegistry` function.

   If terminals have been registered for at least one device type of the instance device type, the `ciCollectDeviceInfo` function gathers information for the terminals found in the table returned by the `ciDeviceInfoRegistry` function.

   For example, consider the following registration:

   ```
   ciDeviceInfoRegisterParams("passive" '(("segments" 1)))
   ```

   Now if you call the following function, the parameters registered for `"R0"` are the ones registered for the `"passive"` device type:

   ```
   ciCollectDeviceInfo(ciGetCellView() '(("R0" inst)))
   ```

   This means that `"R0"` is a device of type `"passive"` and `"resistor"`. You have registered the parameters for `"passive"`, but nothing has been registered for `"resistor"`. As a result, `ciCollectDeviceInfo` no longer uses the default parameter for `"resistor"`.

If an instance has only been registered as a device of type `"resistor"`, `ciCollectDeviceInfo` gathers information for the `"resistor"` parameters, which are the default ones.

```
ciDeviceInfoRegisterParams("resistor" '(("value" 1e-12)))
```

Now, if you calls the following:

```
ciCollectDeviceInfo(ciGetCellView() '(("R0" inst)))
```

Then, the parameters registered for `"R0"` are the ones registered for `"passive"` and `"resistor"` device types. That is, `"R0"` is a device of type `"passive"` and `"resistor"`, and you have registered parameters for `"passive"` and `"resistor"`. As a result, `ciCollectDeviceInfo` gathers information for both `"passive"` and `"resistor"` parameters.

Note: This same reasoning applies to terminals. Also, registering parameters for a device type means that the parameter must exist on all of devices of this type. Use `ciMapParam` if the parameter has different names from device to device, but the same meaning. As a result, you might need to specify that a device type does not have any parameter registered on it. For example, it might be best to avoid registering any parameter for `"passive"` device types. To do this, you can call the following:

```
ciDeviceInfoRegisterParams("passive" '(nil))
```

Now, `ciDeviceInfoGetRegisteredParams("passive")` returns `nil`.

If you instead call the following:

```
ciDeviceInfoRegisterParams("passive" nil)
```

Now, `ciDeviceInfoGetRegisteredParams("passive")` returns the default parameters.

How ciCollectDeviceInfo handles schematic pcells?

By default, `ciCollectDeviceInfo` does not collect information about device inside a submaster view of a Pcell. If it is needed for some Pcell device types, you can register them as device of type `pcellRetrieveSubmaster` (see ciRegisterDevice). In this case, `ciCollectDeviceInfo` collects information about Pcell devices in the submaster cellview if it is found. For detailed information about pcells, see *Virtuoso Parameterized Cell Reference*.

For example, if you have a Pcell iterated device named `R3<1:0>` and its submaster cellview contains two resistors named `R0` and `R1`, then by default (that is, `R3<1:0>` is not registered as a device of type `pcellRetrieveSubmaster`) the following command:

```
ciCollectDeviceInfo(cache '(("/R3<1:0>/R0" inst) ("/R3<1:0>/R1" inst))
?warnUnmapped nil ?warnParamUnfound nil)
```

Returns:

```
(nil devParamNames (
    ("mFactor" 1)
    ("fingerCount" 1)
    ("length" nil)
    ("width" nil)
    ("fingerWidth" nil)
) devTerminalNames ("source" "gate" "drain" "bulk")
cache ci:0x3dff6f30 uniqueNets ("MINUS" "PLUS" "B") devs
((nil termNets
    ("MINUS" "PLUS" "B") name "R3<1:0>/R0"
    dbId db:0x3285849d deviceTypes
    (default) numIter
    2 mFactor 1 fingerCount 1
    length 1.0185e-05 width 2e-06 fingerWidth
    2e-06 source nil gate nil
    drain nil bulk "B" termNet1
    "MINUS" termNet2 "PLUS" termNet3 "B"
    )
) sourceNets nil sourceNetsUnique nil
gateNets nil gateNetsUnique nil drainNets
nil drainNetsUnique nil bulkNets ("B")
bulkNetsUnique ("B")
)
```

If it has been registered as a device of type `pcellRetrieveSubmaster`, that is:

```
ciRegisterDevice("pcellRetrieveSubmaster" '(("gpdk090" "resnsppoly" nil)))
```

Then:

```
ciCollectDeviceInfo(cache '(("/R3<1:0>/R0" inst) ("/R3<1:0>/R1" inst))
?warnUnmapped nil ?warnParamUnfound nil)
```

Returns

```
(nil devParamNames (
    ("mFactor" 1)
    ("fingerCount" 1)
    ("length" nil)
    ("width" nil)
    ("fingerWidth" nil)
) devTerminalNames ("source" "gate" "drain" "bulk")
cache ci:0x3dff6f30 uniqueNets ("MINUS" "PLUS" "B") devs
((nil termNets
```

```
    ("MINUS" "PLUS" "B") name "R3<1:0>/R0"
    dbId db:0x3285869d deviceTypes
    (default) numIter
    2 mFactor 1 fingerCount 1
    length 1.0185e-05 width 2e-06 fingerWidth
    2e-06 source nil gate nil
    drain nil bulk "B" termNet1
    "MINUS" termNet2 "PLUS" termNet3 "B"
    )
    (nil termNets
    ("MINUS" "PLUS" "B") name "R3<1:0>/R1"
    dbId db:0x3285869e deviceTypes
    (default) numIter
    2 mFactor 1 fingerCount 1
    length 1.0185e-05 width 2e-06 fingerWidth
    2e-06 source nil gate nil
    drain nil bulk "B" termNet1
    "MINUS" termNet2 "PLUS" termNet3 "B"
    )
) sourceNets nil sourceNetsUnique nil
gateNets nil gateNetsUnique nil drainNets
nil drainNetsUnique nil bulkNets ("B" "B")
bulkNetsUnique ("B")
)
```

Note: The following SKILL functions also return the Pcell submaster view for the registered device:

```
ciFindObjectInHier(objectFullPathName cache objectType)
ciGetObjectCellView(objectFullPathName cache)
ciGetCellViewForObjectPath(objectFullPathName currentHierPath)
```

Example

```
ciCollectDeviceInfo(ciGetCellView() '(("R0" inst) ("D0" inst)))
=>
list(nil
    'devParamNames ( ("area" nil) ("segments" nil))
    'devTerminalNames ("plus" "minus")
    'cache ci:0x26425c20
    'uniqueNets ("net049" "vdd!" "net070" "net071")
    'devs
        ((nil 'termNets
```

```
        ("net049" "vdd!") 'name "R0"
        'dbId db:0x1f22cd47 'deviceTypes
        ("resistor" "passive") 'numIter 1 'mFactor 1 'fingerCount 1
        'fingerWidth nil 'segments 1 'plus
        "vdd!" 'minus "net049" 'termNet1 "net049"
        'termNet2 "vdd!"
        )
        (nil 'termNets
        ("net070" "net071") 'name "D0"
        'dbId db:0x1f22cd4b 'deviceTypes
        ("passive" "diode") 'numIter 1 'mFactor 1 'fingerCount 1
        'fingerWidth nil 'area 4e-14 plus
        "net070" 'minus "net071" 'termNet1 "net070"
        'termNet2 "net071"
        )
    )
'plusNets ("vdd!" "net070")
'plusNetsUnique ("vdd!" "net070")
'minusNets ("net049" "net071")
'minusNetsUnique ("net049" "net071")
)
```

ciConvertParamsDPLToParams

```
ciConvertParamsDPLToParams(
     l_paramsDPL
     )
     => l_result
```

Description

Converts a disembodied property list of parameter names and values into the list format required for specifying constraint parameters and constraint member parameters. This function is the complement of ciConvertParamsToDPL.

Arguments

l_paramsDPL	The disembodied property list of parameter names and values.

Value Returned

l_result	A list of lists where each sublist contains the parameter name and value.

Example

```
symmConParams = aSymmetryConnstraint->parameters
    (("mirror" 1)
        ("scope" "boundary")
        ("checkWithHalo" nil)
        ("allowedLayers" "N/A")
        ("allowedVias" "N/A")
        ("layerWidths" "N/A")
        ("viaNumCuts" "N/A")
        ("defSpacing" "N/A")
        ("refSpacing" "N/A")
        ("msTolerance" 20.0) <<<< Value to change
        ("tranSpacing" "N/A")
        ("hierarchicalScope" nil)
    )


paramsDPL = ciConvertParamsToDPL(symmConParams)
(nil mirror 1
    scope "boundary"
    checkWithHalo nil
    allowedLayers "N/A"
    allowedVias "N/A"
    layerWidths "N/A"
```

```
    viaNumCuts "N/A"
    defSpacing "N/A"
    refSpacing "N/A"
    msTolerance 20.0
    tranSpacing "N/A"
    hierarchicalScope nil
)


paramsDPL->msTolerance = 33.33

ciConvertParamsDPLToParams(paramsDPL)
    (("mirror" 1)
        ("scope" "boundary")
        ("checkWithHalo" nil)
        ("allowedLayers" "N/A")
        ("allowedVias" "N/A")
        ("layerWidths" "N/A")
        ("viaNumCuts" "N/A")
        ("defSpacing" "N/A")
        ("refSpacing" "N/A")
        ("msTolerance" 33.33) <<<< Value has changed
        ("tranSpacing" "N/A")
        ("hierarchicalScope" nil)
    )
```

ciConvertParamsToDPL

```
ciConvertParamsToDPL(
     l_params
     [ ?filter t_filter ]
     [ ?noTypesSpec g_noTypesSpec ]
     )
     => l_result
```

Description

Converts a constraint parameter list into a disembodied property list of parameter names and values. This function is the complement of ciConvertParamsDPLToParams. Converting the parameter list into a DPL makes it easier to change values in the list.

Arguments

l_params	A list of lists where each sub-list contains the parameter name and value (this is the list format that constraint parameters are specified in).
?filter *t_filter*	Converts only parameters that match the passed text filter. Default: `nil`
?noTypesSpec *g_noTypesSpec*	Specifies whether the passed parameters contains parameter type specifiers or not. Default: `nil`

Value Returned

l_result	The disembodied property list of parameter names and values.

Example

```
symm->parameters
    (("mirror" boolean 1)
    ("scope" enum "boundary")
    ("checkWithHalo" enum "nil")
    ("allowedLayers" string "N/A")
    ("allowedVias" string "N/A")
    ("layerWidths" string "N/A")
```

```
    ("viaNumCuts" string "N/A")
    ("defSpacing" string "N/A")
    ("refSpacing" string "N/A")
    ("msTolerance" float 20.0) <<<< Value to change
    ("tranSpacing" string "N/A")
    ("hierarchicalScope" enumset nil)
)
paramsDPL = ciConvertParamsToDPL(symmConParams) (nil mirror 1
    scope "boundary" checkWithHalo nil allowedLayers "N/A"
    allowedVias "N/A" layerWidths "N/A" viaNumCuts "N/A"
    defSpacing "N/A" refSpacing "N/A" msTolerance 20.0
    tranSpacing "N/A" hierarchicalScope nil)
paramsDPL->msTolerance = 33.33
ciConvertParamsDPLToParams(paramsDPL)
    (("mirror" 1)
    ("scope" "boundary")
    ("checkWithHalo" nil)
    ("allowedLayers" "N/A")
    ("allowedVias" "N/A")
    ("layerWidths" "N/A")
    ("viaNumCuts" "N/A")
    ("defSpacing" "N/A")
    ("refSpacing" "N/A")
    ("msTolerance" 33.33) <<<< Value has changed
    ("tranSpacing" "N/A") ("hierarchicalScope" nil)
)
```

ciConvertToConArg

```
ciConvertToConArg(
     t_guiArg
     )
     => l_result
```

Description

Converts a GUI-friendly parameter name (which can contain spaces) into a legal constraint parameter name (must not contain spaces).

Arguments

t_guiArg	The GUI friendly parameter name to be converted.

Value Returned

l_result	A legal constraint parameter name.

Example

```
ciConvertToConArg(stringToSymbol("Add Guard Ring"))
=> "addGuardRing"
```

ciCreateGuardRing

```
ciCreateGuardRing(
     g_cache
     g_modgen
     l_args
     )
     => t_result
```

Description

Default function used by the Rapid Analog Prototype constraint generators for creating guard rings. If the modgen is already associated with a guard ring, `ciCreateGuardRing` updates the parameters of the existing guard ring.

Arguments

g_cache	The constraints cache.
g_modgen	The modgen constraint for which the guardring is being created.
l_args	A disembodied property list of the constraint generator arguments that contains parameters that will control how the default guardring is created.

Value Returned

t_result	The guard ring constraint.

Example

```
guardRingArgs = list( nil stringToSymbol("MPP") "Metal3"
                stringToSymbol("Net")     "net101"
                stringToSymbol("Type")    "ring"
                stringToSymbol("Shape")   "rectilinear"
                stringToSymbol("Spacing") 0.25
                )
modgen = ciConFind(cache "myModgen")
guardRing = ciCreateGuardRing(cache modgen guardRingArgs)
```

ciCreateModgen

```
ciCreateModgen(
     g_cache
     l_instsNetsPins
     s_structType
     l_args
     [ ?createModgenTemplate g_createModgenTemplate ]
     )
     => templateID / constraintID
```

Description

A constraint generation utility function, which simplifies the process of generating modgens for specified structure types. This function calls expressions to create the modgen device pattern and guard ring that have been pre-registered for the specified structure type. For example, `DiffPair` and `CurrentMirror`.

This function will evaluate modgen pattern expression that have been pre-registered for the specified structure type as well as handling device abutment, what type of guardringto add and whether to add dummies or not.

It is also possible to register alternative modgen pattern expression using `ciSetStructGeneratorExpressions()`.

In addition, the argument list supplied to `ciRegisterConstraintGenerator` supports separators to make the popup constraint generator options dialog easier to read. This will be achieved by using the `'separator` argument type, as shown in the example below:

```
args = ( …
list("Guard Ring Options" 'separator)
…)
```

The argument list also supports expandable/collapsable groups of arguments. This will be achieved by using `'beginExpandedOptions`/`'endExpandedOptions` argument types, as shown in the example below:

```
args = ( …
list("Guard Ring Options" 'beginExpandedOptions)
… guard ring args specified between begin/end …
list("Guard Ring Options" 'endExpandedOptions)
…)
```

In addition, it supports the `'hide` parameter (optional) on each argument to control the visibility of the argument on the dialog. This will allow you to trim down and simplify the dialogs where default values of arguments are deemed sufficient and/or you should be prevented from altering these values, as shown in the example below:

```
args = ( …
list("Guard Ring Type" 'enum list("none" "ring" "pane" "ring") 'hide)
)
```

Arguments

g_cache — The constraints cache.

l_instNetsPins — A list of sublists where each sublist contains object names and types. The objects in this case will be insts, which are to become members of the created modgen.

s_structType — Specifies the type of structure the modgen is being created for. The predefined types are: `DiffPair`, `CurrentMirror`, `CascodedCurrentMirror`, `LargeMfactor`, and `PassiveDeviceArray`.

l_args — A disembodied property list of constraint generator argument values for this structure type.

`?createModgenTemplates` *g_createModgenTemplates*

This is an optional argument that defaults to `t` and controls whether the modgen is created inside a template constraint or not. This option can only be used if a template has been defined for the structType using `ciTemplateCreateDefinition()` with ?params set to `ciGetStructArgs(structType)` so that the template parameters match the args registered for the struct type.

Value Returned

templateID — The ID of the template generated when the `createModgenTemplate` Boolean argument defaults to `t`.

constraintID	The ID of the constraint generated when the `createModgenTemplate` Boolean argument is set to `nil`.

Examples

Example 1

```
ciRegisterConstraintGenerator(list(nil
        'name   "Module (Diff Pair)"
        'description    "Generate Modgen Constraint for a Differential Pair"
        'expression     "ciCreateModgen(cache instsNetsPins 'DiffPair args)" <<<<
        'addToToolbar   t
        'iconName       "DiffPair"
        'args           "ciGetStructArgs('DiffPair)"
        'menu           "Rapid Analog Prototype"
)
)
```

This function is typically used when registering a new constraint generator.

Example 2

As shown in the example below, the new Module (Current Mirror) constraint generator calls the new `ciCreateModgen()` function for generating the Current Mirror specific modgen:

```
ciRegisterConstraintGenerator( list(nil
    'name "Module (Current Mirror)"
    'description "Generate Modgen Constraint for a Current Mirror"
    'expression "ciCreateModgen(cache instsNetsPins 'CurrentMirror args
    ?createModgenTemplate t)"
    'addToToolbar t
    'iconName "CurrentMirror"
    'args "ciGetStructArgs(`CurrentMirror)"
)
```

where, `ciGetStructArgs('CurrentMirror)` is a new CP function that returns a pre-registered list of constraint generator arguments for a CurrentMirror. A new CP registration function, `ciSetStructArgs(<structSymbolName> <argsList>)` allows you to add/modify/delete your own specific arguments for their structures. Functions provided for adding/updating/deleting a specific argument in the argument list are mentioned below:

```
ciAddStructArg(<structSymbolName> <argDef> ?afterArgName nil) ;;; add a new arg to
end of args or after specified argnName

argDef = ciGetStructArg(<structSymbolName> <argName>) ;;; get a specified argument

ciReplaceStructArg(<structSymbolName> <argName> <argDef>) ;;; replace a specified
argument

ciDeleteStructArg(<structSymbolName> <argName> ) ;;; delete a specified argument
```

For example, to change the default for the “Add GuardRing” Boolean option from `nil` to `t`:

```
addGuardRingArg = ciGetStructArg('CurrentMirror “Add GuardRing”) ;;; ("Add
GuardRing" bool nil)
addGuardRingArg = list(car(addGuardRingArg) cadr(addGuarRingArg) t)
ciReplaceStructArg(‘CurrentMirror “Add GuardRing” addGuardRingArg)
```

ciCreateModgenDummy

```
ciCreateModgenDummy(
     g_cache
     d_refDev
     x_row
     x_col
     g_abut
     [ ?dummyNetToUse x_dummyNetToUse ]
     [ ?orient s_orient ]
     [ ?dummyParams l_dummyParams ]
     [ ?extraParams l_extraParams ]
     )
     => l_result
```

Description

A utility function for creating a modgen member parameter for specifying a dummy device.

Arguments

g_cache	The constraints cache.
d_refDev	The `dbInstId` for the reference device to be used for the creation of the dummy.
x_row	The modgen row index for the dummy device.
x_col	The modgen column index for the dummy device.
g_abut	Controls whether the dummy device should be abutted or not.
`?dummyNetToUse` *x_dummyNetToUse*	
	Optional net index of the dummy net to use. It defaults to `1`.
`?orient` *s_orient*	Optional orientation for the dummy. It defaults to `'R0`.
`?dummyParams` *l_dummyParams*	Optional list of dummy parameters. It defaults to `nil`. When set to `nil`, it sets the `length`, `width`, `fingerCount`, and `fingerWidth` parameters based on the passed `refDev` parameter values.

`?extraParams` *`l_extraParams`*	Optional list of modgen member parameters. It defaults to `nil`. When set to `t`, it allows additional member parameters to be specified on the dummy. For example, vertical or horizontal spacing.

Value Returned

`l_result`	A list of lists where each sublist is a modgen member parameter for the dummy device.

Example

```
cv = geGetEditCellView()
refDev = dbFindAnyInstByName(cv "NM1")


ciCreateModgenDummy(cache refDev 3 5 t ?dummyNetToUse 2 ?orient R270)
("gpdk045/nmos2v/layout" master
     (("row" 3)
     ("col" 5)
     ("abutment" t)
     ("dummyNetToUse" 2)
     ("orient" 'R270)
     ("dummyParams" "((w 1.000000e-06) (l 5.400000e-07) (fingers 2)
        (fw 5.000000e-07))")
     )
)
```

ciCreateModgenForStructure

```
ciCreateModgenForStructure(
     g_cache
     l_instsNetsPins
     s_structType
     l_args
     )
     => l_modgen
```

Description

A function performs the same basic function as ciCreateModgen, but does not have the createModgenTemplate argument.

Arguments

g_cache	The constraints cache.
l_instsNetsPins	A list of sublists where each sublist contains object names and types. The objects in this case will be insts which are to become members of the created modgen.
s_structType	Specifies the type of structure the modgen is being created for.
l_args	A disembodied property list of constraint generator argument values for this structure type.

Value Returned

l_modgen	A list containing the generated modgen constraint and optionally a guardring constraint.

Example

Typically used when registering a new constraint generator that does not require template constraints to be created:

```
ciRegisterConstraintGenerator(list(nil
    'name   "Module (Diff Pair)"
    'description    "Generate Modgen Constraint for a Differential Pair"
    'expression     "ciCreateModgenForStructure(cache instsNetsPins 'DiffPair
    args)"
```

```
    'addToToolbar   t
    'iconName       "DiffPair"
    'args           "ciGetStructArgs('DiffPair)"
    'menu           "Rapid Analog Prototype"
    )
)
```

ciCreateRoutingLayerEnumString

```
ciCreateRoutingLayerEnumString(
     x_defLayerIndx
     )
     => t_result
```

Description

Utility function for creating layer name enums, which will retrieve the list of layer names from the technology file and make layerNumber the default by adding the associated layer name to the end of the list.

Arguments

x_defLayerIndx	The layer index to be used as the default layer in the string list.

Value Returned

t_result	A string containing the list of routing layers where the routing layer for `defLayerIndx` is also appended to the string so that it becomes the default routing layer.

Example

```
ciCreateRoutingLayerEnumString(2)
=> list("Metal1" "Metal2" "Metal3" "Metal4" "Metal5" "Metal6" "Poly" "Metal2")
```

ciDeleteClusterMembersWithinModgens

```
ciDeleteClusterMembersWithinModgens(
     g_cache
     )
     => t
```

Description

A utility function for removing devices from cluster constraints if those devices are also members of a modgen.

Arguments

g_cache	The current cache.

Value Returned

`t`	Returns `t` if any device is removed from a cluster constraint.

Example

```
cluster = ciConCreate(cache 'cluster ?members '(("NM1" inst) ("NM2" inst) ("NM3"
inst) ("NM4" inst)))
modgen = ciConCreate(cache 'modgen ?members '(("NM1" inst) ("NM2" inst)))
ciDeleteClusterMembersWithinModgens(cache)
Deleting member "NM1" from cluster "Constr_4" since it is also a member of modgen:
"Constr_5"
Deleting member "NM2" from cluster "Constr_4" since it is also a member of modgen:
"Constr_5"
cluster->members
(("NM3" inst nil)
("NM4" inst nil)
)
```

ciDeleteGuardRing

```
ciDeleteGuardRing(
     g_modgen
     )
     => t / nil
```

Description

If the passed modgen constraint has a Guard Ring constraint associated with it then delete it.

Arguments

g_modgen	The modgen constraint whose Guard Ring constraint is to be deleted.

Value Returned

`t`	The Guard Ring constraint is deleted.
`nil`	Command failed.

Example

```
guardRingArgs = list( nil stringToSymbol("MPP")     "Metal3"
                stringToSymbol("Net")     "net101"
                stringToSymbol("Type")    "ring"
                stringToSymbol("Shape")   "rectilinear"
                stringToSymbol("Spacing") 0.25
)

modgen = ciConFind(cache "myModgen")
guardRing = ciCreateGuardRing(cache modgen guardRingArgs)
ciDeleteGuardRing(modgen)
```

ciDeleteStructArg

```
ciDeleteStructArg(
     s_structType
     t_argName
     )
     => t / nil
```

Description

Deletes the named argument from the registered argument list for the passed `structType`. See also, ciGetStructArgs, ciSetStructArgs, ciRegexReplaceStructArgs, ciAddStructArg, ciReplaceStructArg, and ciGetStructArg.

Arguments

s_structType	The structure type.
t_argName	The name of the argument to delete.

Value Returned

`t`	The command has succeeded.
`nil`	Command failed.

Example

```
ciDeleteStructArg('GuardRing "Guard Ring Settings")
```

This function removes the Guard Ring options expander button.

ciDeleteSymmetriesWithinModgens

```
ciDeleteSymmetriesWithinModgens(
     g_cache
     )
     => t / nil
```

Description

Removes the devices from symmetry constraints if those devices are also members of a modgen.

Arguments

g_cache	The constraint cache ID.

Value Returned

`t`	Returns `t` if any device is removed from the symmetry constraint.
`nil`	Command failed.

Example

```
symmetry = ciConCreate(cache 'symmetry ?members '(("NM1" inst) ("NM4" inst)))
modgen = ciConCreate(cache 'modgen ?members '(("NM1" inst) ("NM2" inst)))
ciDeleteSymmetriesWithinModgens(cache)
```

```
Deleting symmetry constraint "Constr_6" ((("NM1" inst nil) ("NM4" inst nil))) since
some of its members are contained within a modgen: ("Constr_7") [(("NM1" inst)
("NM2" inst))]
```

```
=> t
```

ciExpandAndRepeatName

```
ciExpandAndRepeatName(
     t_name
     x_numRepetitions
     s_type
     )
     => t / nil
```

Description

Expands iterated device names and repeats the expanded name the required number of times. The repetition is necessary where the iterated device is also M factored.

Arguments

t_name	The iterated device name.
x_numRepetitions	The number of times the iterated name is to be repeated.
s_type	The constraint member type for the passed name.

Value Returned

`t`	The iterated device names were expanded and repeated the required number of times.
`nil`	The iterated device names could not be expanded and repeated.

Example

```
ciExpandAndRepeatName("MN1<3:5>" 2 'inst)
=>'("MN1<3>" "MN1<3>" "MN1<4>" "MN1<4>" "MN1<5>" "MN1<5>")
```

ciExtractRowNumber

```
ciExtractRowNumber(
     t_rowString
     [ ?rowPrefix t_rowPrefix ]
     )
     => t / nil
```

Description

Extracts the row number from a string of the form `row` followed by a `rowNumber`.

Arguments

t_rowString	Specifies the string from which to extract the row number.
?rowPrefix *t_rowPrefix*	Specifies the default `rowString` prefix.

Value Returned

t	Returns the extracted row number.
nil	Returns `nil` if the `rowString` does not start with the `rowPrefix` string.

Examples

The following examples illustrate different uses of the `ciExtractRowNumber` SKILL command:

```
ciExtractRowNumber("row3")
=> 3
ciExtractRowNumber("col4")
=> nil
ciExtractRowNumber("col4" ?rowPrefix "col")
=> 4
```

ciFindDeviceArraysForDev

```
ciFindDeviceArraysForDev(
     d_dev
     )
     => t / nil
```

Description

This function is used for extracting the list of devices which have the same master and are connected in series or parallel chains. This can be used to identify series/parallel resistors/capacitors. The bulk connections of these devices are ignored.

Arguments

d_dev	Specifies the database ID of the device for which the series/parallel connections are required.

Value Returned

`t`	Returns the list of series/parallel devices connected to the device for which the connections are required.
`nil`	Returns `nil` if no series/parallel devices are connected.

Example

```
res = dbFindAnyInstByName(geGetEditCellView() "res1")
ciFindDeviceArraysForDev(res)~>name
'("res1" "res2" "res3" "res4")
```

ciGenerateArrayChannelDesc

```
ciGenerateArrayChannelDesc(
     g_cache
     l_devInfo
     l_pattern
     args
     )
     => t_channelDesc
```

Description

Generates the Pin To Trunk routing information for Passive Device Array structures. This function is called as part of the constraint generation process in the routing expression for Passive Device Array structures which is registered by `ciSetStructGeneratorExpressions('PassiveDeviceArray ...)`.

Arguments

g_cache	Specifies the constraints cache.
l_devInfo	Specifies the constraint generator device information disembodied property list as returned by `ciCollectDeviceInfo`.
l_pattern	Specifies the modgen member pattern which is a list of sublists where each sublist represents a row in the modgen. The sublist contains the device name, type, and optionally the parameters.
args	Specifies the constraint generator argument values.

Value Returned

t_channelDesc	Returns the Pin To Trunk routing description for a Passive Device Array structure.

Example

```
ciSetStructGeneratorExpressions('PassiveDeviceArray
    make_ciStructGeneratorExpressions(
    ?pattern   "ciGenerateBestFitPattern(devInfo args)"
    ?routing   "when(pattern && args->Route ciGenerateArrayChannelDesc(cache
                devInfo pattern args))"
```

```
    ?guardRing guardRingExpr
))
```

ciGenerateBestFitPattern

```
ciGenerateBestFitPattern(
     l_devInfo
     args
     [ ?expandInsts g_expandInsts ]
     [ ?forcedRows x_forcedRows ]
     )
     => l_pattern
```

Description

Generates the modgen member pattern (interdigitation) information for a Passive Device Array structure. This function is called as part of the constraint generation process in the pattern expression for Passive Device Array structures that are registered by `ciSetStructGeneratorExpressions('PassiveDeviceArray ...)`.

Arguments

l_devInfo	Specifies the constraint generator device information disembodied property list as returned by `ciCollectDeviceInfo`.
args	The constraint generator argument values.

Value Returned

l_pattern	The modgen member pattern which is a list of sublists where each sublist represents a row in the modgen. The sublist contains the device name, type, and optionally the parameters.
`?expandInsts` *g_expandInsts*	Optional Boolean argument that defaults to `t` and controls whether devices should be expanded based on their mfactor. The device names of mfactored devices must appear in the pattern as many times as their mfactor.
`?forcedRows` *x_forcedRows*	Optional integer parameter that can be used to force the number of rows to the specified value. The default of `-1` means that the number of rows will be chosen automatically.

Example

```
ciSetStructGeneratorExpressions('PassiveDeviceArray
make_ciStructGeneratorExpressions(
    ?pattern   "ciGenerateBestFitPattern(devInfo args)"
    ?guardRing guardRingExpr
))
```

ciGenerateCascodedCurrentMirrorChannelDesc

```
ciGenerateCascodedCurrentMirrorChannelDesc(
     g_cache
     l_devInfo
     l_pattern
     args
     )
     => t_channelDesc
```

Description

This function is called as part of the constraint generation process and generates the Pin To Trunk routing information for a Cascoded Current Mirror structure. This function is called in the routing expression for Cascoded Current Mirrors structures which is registered by `ciSetStructGeneratorExpressions('CascodedCurrentMirror ...)`.

Arguments

g_cache	The Constraints cache.
l_devInfo	Specifies the constraint generator device information disembodied property list as returned by `ciCollectDeviceInfo`.
l_pattern	The modgen member pattern which is a list of sublists where each sublist represents a row in the modgen. The sublist contains the device name, type, and optionally the parameters.
args	The constraint generator argument values.

Value Returned

t_channelDesc	Returns the Pin To Trunk routing description for a Cascoded Current Mirror.

Example

```
ciSetStructGeneratorExpressions('CascodedCurrentMirror
make_ciStructGeneratorExpressions(
    ?pattern   "ciGenerateCascodedCurrentMirrorPattern(devInfo args)"
    ?routing   "when(pattern && args->Route
         ciGenerateCascodedCurrentMirrorChannelDesc(cache devInfo pattern args))"
```

```
    ?guardRing guardRingExpr
))
```

ciGenerateCascodedCurrentMirrorPattern

```
ciGenerateCascodedCurrentMirrorPattern(
     l_devInfo
     args
     )
     => l_pattern
```

Description

This function is called as part of the constraint generation process and generates the modgen member pattern (interdigitation) information for a Cascoded Current Mirror structure. This function is called in the pattern expression for Cascoded Current Mirror structures which is registered by `ciSetStructGeneratorExpressions('CascodedCurrentMirror ...)`.

Arguments

l_devInfo	Specifies the constraint generator device information disembodied property list as returned by `ciCollectDeviceInfo`.
args	The constraint generator argument values.

Value Returned

l_pattern	The modgen member pattern which is a list of sublists where each sublist represents a row in the modgen. The sublist contains the device name, type, and optionally the parameters.

Example

```
ciSetStructGeneratorExpressions('PassiveDeviceArray
make_ciStructGeneratorExpressions(
    ?pattern   "ciGenerateBestFitPattern(devInfo args)"
        ?guardRing guardRingExpr
))
```

ciGenerateCurrentMirrorChannelDesc

```
ciGenerateCurrentMirrorChannelDesc(
     g_cache
     l_devInfo
     l_pattern
     args

     )
     => t_channelDesc
```

Description

This function is called as part of the constraint generation process and generates the Pin To Trunk routing information for a Current Mirror structure. This function is called in the routing expression for Current Mirror structures, which is registered by `ciSetStructGeneratorExpressions('CurrentMirror ...)`.

Arguments

g_cache	The constraints cache.
l_devInfo	The constraint generator device information disembodied property list as returned by ciCollectDeviceInfo.
l_pattern	The modgen member pattern, which is a list of sublists where each sublist represents a row in the modgen. The sublist contains the device name, type, and parameters (optionally).
args	The constraint generator argument values.

Value Returned

t_channelDesc	Returns the Pin To Trunk routing description for a Current Mirror structure.

Example

```
ciSetStructGeneratorExpressions('CurrentMirror make_ciStructGeneratorExpressions(
    ?pattern   "ciGenerateCurrentMirrorPattern(devInfo args)"
    ?routing   "when(pattern && args->Route
    ciGenerateCurrentMirrorChannelDesc(cache devInfo pattern args))"
```

```
    ?guardRing guardRingExpr
))
```

ciGenerateCurrentMirrorPattern

```
ciGenerateCurrentMirrorPattern(
     l_devInfo
     args

     )
     => l_pattern
```

Description

This function is called as part of the constraint generation process and generates the modgen member pattern (interdigitation) information for a Current Mirror structure. This function is called in the pattern expression for Current Mirror structures, which is registered by `ciSetStructGeneratorExpressions('CurrentMirror ...)`.

Arguments

l_devInfo	The constraint generator device information disembodied property list as returned by ciCollectDeviceInfo.
args	The constraint generator argument values.

Value Returned

l_pattern	The modgen member pattern, which is a list of sublists where each sublist represents a row in the modgen. The sublist contains the device name, type, and parameters (optional).

Example

```
ciSetStructGeneratorExpressions('CurrentMirror make_ciStructGeneratorExpressions(
    ?pattern   "ciGenerateCurrentMirrorPattern(devInfo args)"
    ?guardRing guardRingExpr
))
```

ciGenerateDiffPairChannelDesc

```
ciGenerateDiffPairChannelDesc(
     g_cache
     l_devInfo
     l_pattern
     args
     )
     => t_channelDesc
```

Description

This function is called as part of the constraint generation process and generates the Pin To Trunk routing information for a Diff Pair structure. This function is called in the routing expression for Diff Pair structures, which is registered by `ciSetStructGeneratorExpressions('DiffPair ...)`.

Arguments

g_cache	The constraints cache.
l_devInfo	Specifies the constraint generator device information disembodied property list as returned by `ciCollectDeviceInfo`.
l_pattern	The modgen member pattern which is a list of sublists where each sublist represents a row in the modgen. The sublist contains the device name, type, and optionally the parameters.
args	The constraint generator argument values.

Value Returned

t_channelDesc	Returns the Pin To Trunk routing description for a Diff Pair structure.

Example

```
ciSetStructGeneratorExpressions('DiffPair make_ciStructGeneratorExpressions(
    ?pattern   "ciGenerateDiffPairPattern(devInfo args)"
    ?routing   "when(pattern && args->Route ciGenerateDiffPairChannelDesc(cache
    devInfo pattern args))"
    ?guardRing guardRingExpr
))
```

ciGenerateDiffPairPattern

```
ciGenerateDiffPairPattern(
     l_devInfo
     args
     )
     => l_pattern
```

Description

This function is called as part of the constraint generation process and generates the modgen member pattern (interdigitation) information for a Differential Pair structure. This function is called in the pattern expression for Differential Pair structures, which is registered by `ciSetStructGeneratorExpressions('DiffPair ...)`.

Arguments

l_devInfo	The constraint generator device information disembodied property list as returned by `ciCollectDeviceInfo`.
args	The constraint generator argument values.

Value Returned

l_pattern	The modgen member pattern, which is a list of sublists where each sublist represents a row in the modgen. The sublist contains the device name, type, and optionally the parameters.

Example

```
ciSetStructGeneratorExpressions('DiffPair make_ciStructGeneratorExpressions(
    ?pattern   "ciGenerateDiffPairPattern(devInfo args)"
    ?guardRing guardRingExpr
))
```

ciGenerateLargeMfactorPattern

```
ciGenerateLargeMfactorPattern(
     l_devInfo
     args
     )
     => l_pattern
```

Description

This function is called as part of the constraint generation process and generates the modgen member pattern (interdigitation) information for a Large Mfactor structure. This function is called in the pattern expression for Large Mfactor structures, which is registered by `ciSetStructGeneratorExpressions('LargeMfactor ...)`.

Arguments

g_modgen	The constraint generator device information disembodied property list as returned by `ciCollectDeviceInfo`.
args	The constraint generator argument values.

Value Returned

l_pattern	The modgen member pattern which is a list of sublists where each sublist represents a row in the modgen. The sublist contains the device name, type, and optionally the parameters.

Example

```
ciSetStructGeneratorExpressions('LargeMfactor make_ciStructGeneratorExpressions(
    ?pattern   "ciGenerateLargeMfactorPattern(devInfo args)"
    ?guardRing guardRingExpr
))
```

ciGetGuardRing

```
ciGetGuardRing(
     g_modgen
     )
     => g_guardRing
```

Description

Returns the guard ring constraint associated with the passed modgen constraint, if it has one.

Arguments

g_modgen	The modgen constraint for which the guardring has to be determined.

Value Returned

g_guardRing	Returns the guard ring associated with the passed modgen or `nil` if there is none.

Example

```
guardRingArgs = list( nil stringToSymbol("MPP")      "Metal3"
    stringToSymbol("Net")      "net101"
    stringToSymbol("Type")     "ring"
    stringToSymbol("Shape")    "rectilinear"
    stringToSymbol("Spacing") 0.25
)
modgen = ciConFind(cache "myModgen")
guardRing = ciCreateGuardRing(cache modgen guardRingArgs)
ciGetGuardRing(modgen) == guardRing
t
```

ciGetGuardRingMPPName

```
ciGetGuardRingMPPName(
     d_dev
     )
     => t_mppName
```

Description

This function returns the registered technology file Guard Ring MPP name for the passed device based on the device type (`nfet` or `pfet`).

The Guard Ring MPP name lists are by default registered as follows:

- `ciMapParam("GuardRing.mpp.pfet" '("P-Tap"))`
- `ciMapParam("GuardRing.mpp.nfet" '("N-Tap"))`

The "`pfet`" and "`nfet`" device name lists should be registered with `ciRegisterDevice` and this is normally loaded at PDK load time.

Arguments

d_dev	Database ID of the device.

Value Returned

t_mppName	The technology file MPP name string for this type of device ("`nfet`" or "`pfet`").

Examples

```
ciGetGuardRingMPPName("NM1")
=> "N-Tap"
ciGetGuardRingMPPName("PM1")
=> "P-Tap"
```

ciGetParamValFromParameters

```
ciGetParamValFromParameters(
     t_paramName
     l_params
     [ ?valPos x_valPos ]
     )
     => l_val / nil
```

Description

Retrieves the specified parameter value from a constraint parameter list. It can be used to retrieve values of constraint parameters or constraint member parameters.

Arguments

t_paramName	The name of the parameter for which the value needs to be found.
l_params	The constraint parameter list from which the parameter value is to be found.
?valPos *x_valPos*	The position of the value with the parameter sub-lists. Default is 3.

Value Returned

l_val	The value of the parameter.
nil	Command failed.

Example

```
symmCon = ciConFind("mySymmetryConstraint")
ci:0x27a175f0
symmCon->parameters
    (("mirror" boolean 1)
        ("scope" enum "boundary")
        ("checkWithHalo" enum "nil")
        ("allowedLayers" string "N/A")
        ("allowedVias" string "N/A")
        ("layerWidths" string "N/A")
        ("viaNumCuts" string "N/A")
        ("defSpacing" string "N/A")
        ("refSpacing" string "N/A")
        ("msTolerance" float 20.0)
        ("tranSpacing" string "N/A")
```

```
        ("hierarchicalScope" enumset nil)
)
ciGetParamValFromParameters("msTolerance" symmCon->parameters)
20.0
```

ciGetRoutingLayer

```
ciGetRoutingLayer(
     t_layerName
     )
     => d_layerId / nil
```

Description

Retrieves the named routing layer from the technology file associated with the current window. See also, ciGetTechFile, ciGetRoutingLayers, techGetLayerNum, techGetLayerMaskNumber, techFindLayer, and techGetLayerFunctions.

Arguments

l_layerNames	The name of the layer.

Value Returned

d_layerId	The database ID of the layer.
nil	The database ID of the layer could not be returned.

Example

```
metal1 = ciGetRoutingLayer("Metal1")
metal1->number
30
```

ciGetRoutingLayers

```
ciGetRoutingLayers(
     )
     => l_layerNames
```

Description

Retrieves the names of all the routing layers from the technology file associated with the current window. See also ciGetTechFile, ciGetRoutingLayer, techGetLayerNum, techGetLayerMaskNumber, techFindLayer, and techGetLayerFunctions.

Arguments

None

Value Returned

l_layerNames	List of routing layer names.

Example

```
ciGetRoutingLayers()

=> ("Metal1" "Metal2" "Metal3" "Metal4" "Metal5"
    "Metal6" "Metal7" "Metal8" "Metal9" "Metal10"
    "Metal11" "Poly"
   )
```

ciGetRule

```
ciGetRule(
     t_layerName
     t_ruleName
     g_defVal
     )
     => g_ruleValue / g_defVal
```

Description

Retrieves the required rule value from the technology file. If the rule cannot be found, then the default value is returned.

Arguments

t_layerName	The layer name to get the rule value for.
t_ruleName	The name of the rule.
g_defVal	A default value in case the rule is not found. If a default is not specified then an error will be reported if the rule is not found.

Value Returned

g_ruleValue	Retrieves the named rule on the named layer.
g_defVal	Returns the default value.

Examples

- Retrieves the named rule on the named layer.

  ```
  ciGetRule("Metal1" "minWidth" 0.1)
  => 0.06
  ```

- Returns the default value.

  ```
  ciGetRule("Metal1" "minWidth" 0.123)
  => 0.123
  ```

- Returns the default value.

  ```
  ciGetRule("Metal2" "minWidth" 0.123)
  => 0.3
  ```

ciGetStructArg

```
ciGetStructArg(
     s_structType
     t_argName
     )
     => l_argDef
```

Description

Retrieves the named structure argument for the specified structure type. See also, ciSetStructArgs, ciGetStructArgs, ciReplaceStructArg, ciAddStructArg, and ciDeleteStructArg.

Arguments

s_structType	The structure type from which the named argument is retrieved.
t_argName	The name of the argument to retrieve.

Value Returned

l_argDef	The registered argument for *argName*.

Example

```
ciGetStructArg('DiffPair "Gate Width")
("Gate Width" 'float "ciGetRule(cadr(ciGetRoutingLayers()) \"minWidth\"
0.1)*ciGetStructPDKMult('DiffPair)")
```

ciGetStructArgs

```
ciGetStructArgs(
     s_structType
     )
     => l_argDef
```

Description

Retrieves all the structure arguments for the specified structure type. See also, ciSetStructArgs, ciGetStructArg, ciReplaceStructArg, ciAddStructArg, and ciDeleteStructArg.

Arguments

s_structType	The structure type for which all arguments are to be retrieved.

Value Returned

l_argDef	The list of registered arguments for the passed structure type.

Example 1

```
ciGetStructArgs('DiffPair)
    (("Style" 'enum "\"Auto row1 row2symmetric MultiRowMatched\"")
        ("Add Dummies" 'bool nil)
        ("Abut All" 'bool t)
        ("Guard Ring" 'separator)
        ("Add GuardRing" 'bool nil)
        ("Settings" 'beginExpandedOptions nil)
        ("Type" 'enum "\"ring pane\"")
        ("Shape" 'enum "\"rectangular rectilinear\"")
....
```

Example 2

```
ciGetStructArgs(‘Dummies) ->
list( list("Add Dummies“ `bool nil))
ciGetStructArgs(‘Abutment) ->
list( list("Abut All" `bool t))
ciGetStructArgs(‘GuardRing) ->
```

```
list(
    list("Guard Ring" `separator) ;;; Adds a separator into the GUI -- Guard Ring
    ----------
    list("Add Guard Ring" `bool nil)
    list("Settings" `beginExpandedOptions nil) ;; Adds an expander >>> into the GUI
    list("Type" `enum "\"ring pane\"")
    list("Shape" `enum "\"rectangular rectilinear\"")
    list("Net" `enum "buildString(ciGetMembersOfType(instsNetsPins 'net
    ?includeType nil))")
    list("Spacing" `float "0.0")
    list("MPP" `enum
    "strcat(buildString(techGetMPPTemplateNames(ciGetTechFile())) \" \"
        ciGetGuardRingMPPName(car(ciInstsNetsPinsToDevInfo(cache instsNetsPins)-
        >devs)->dbId))")
    list("Fluid Guard Ring" `separator) ;;; Adds a separator into the GUI -- Fluid
    Guard Ring ----------
    list("Use Fluid" `bool nil)
    list("Device" `enum "buildString(ciGetMembersOfType(instsNetsPins 'inst
    ?includeType nil))")
    list("Width" `float0.0)
    list("Use Min DRC For Spacing" `bool nil)
    list("Settings" `endExpandedOptions nil)
```

As shown above, `ciGetStructPDKMult(structType)` returns a PDK dependent multiplier, which can be applied to the values returned by `ciGetRule()`. This function returns `1.0` when called with `structType` set to default.

In addition, different multipliers can be set for different `structTypes` and PDKs using `ciSetStructPDKMult(structType pdkName multiplier)`, for example, `ciSetStructPDKMult('CurrentMirror "gpdk045" 1.0)`.

Note: `ciGetStructPDKMult()` dynamically determines the PDK name when the expression is evaluated.

ciGetStructGeneratorExpressions

```
ciGetStructGeneratorExpressions(
     s_structType
     )
     => r_expressions
```

Description

Retrieves the constraint generator expressions registered for the passed structure type. The generator expressions are returned within a `ciStructGeneratorExpressions` def struct. These expressions are evaluated by the `ciCreateModgen` SKILL function during modgen creation and modification. See also, ciSetStructGeneratorExpressions, ciListStructGeneratorExpressions, and ciCreateModgen.

Arguments

s_structType	The structure type from which the expressions are retrieved.

Value Returned

r_expressions	A `ciStructGeneratorExpressions` def struct containing the structure generator expressions.

Example

```
ciGetStructGeneratorExpressions('CurrentMirror)~>??
(
 pattern   "ciGenerateCurrentMirrorPattern(devInfo args)"
 guardRing "when(args->\"Add GuardRing\" ciCreateGuardRing(cache modgen args))"
 modgenPreCreateOrModify  "modgenParams = ciMergeParams(modgenParams
list(list(\"mergeLayer\"  args->\"Merge Layer\")))"
 modgenPostCreateOrModify "reorderModgenMembersByDeviceMapping(modgen args-
>\"Device Mapping\")"
)
```

For a complete description of these fields, see ciSetStructGeneratorExpressions.

ciGetStructPDKMult

```
ciGetStructPDKMult(
     s_structType
     )
     => x_pdkMult
```

Description

Retrieves the PDK specific multiplier for the passed structure type. The PDK name is determined from the technology file associated with the current window. If multiplier has been registered then the default value of `1.0` will be returned. The PDK multipliers are typically used within constraint generator argument expressions to scale the values in a PDK independent way. See also, ciSetStructPDKMult.

Arguments

s_structType	The structure type from which the PDK multiplier are retrieved.

Value Returned

x_pdkMult	The PDK multiplier for the passed structure type.

Examples

```
ciGetStructPDKMult('DiffPair)
=>1.25
ciGetStructPDKMult('CurrentMirror)
=>1.0
```

ciGUIArgsToConArgs

```
ciGUIArgsToConArgs(
     l_guiArgs
     )
     => t_conArgs
```

Description

A simple wrapper around `ciConvertToConArg` for converting disembodied property lists of GUI args to a list of list of lists where each sublist is a constraint parameter list.

Arguments

l_guiArgs	Specifies the disembodied property list of GUI arguments.

Value Returned

l_conArgs	Returns the list of lists where each sublist is a constraint parameter list.

Example

```
ciGUIArgsToConArgs(list(nil stringToSymbol("Gate Layer") "Metal1"
stringToSymbol("Gate Width") 0.1))
=> list(list("gateLayer" "Metal1") list("gateWidth" 0.1))
```

ciHighestLevelNet

```
ciHighestLevelNet(
     t_lowerLevelNetName
     )
     => l_higherLevelNetInfo
```

Description

This function returns the `dbId` of the highest level net associated with the passed lower level hierarchical net name. If the net is local to the cellview, which is contained within then the function returns info on that net. See also ciResolveNet.

Arguments

t_lowerLevelNetName	The name of the lower level net.

Value Returned

l_higherLevelNetInfo	A disembodied property list containing the hierarchical path to the higher level net and the `dbId` of the higher level net.

Example

```
netInfo = ciHighestLevelNet("/I60/IN")
netInfo~>??
(nil hierPath "" net db:0x2034f51c)
netInfo->net->name
"REF"
```

ciListStructGeneratorExpressions

```
ciListStructGeneratorExpressions(
     )
     => l_structureExpressions
```

Description

Returns a list of all the registered structure generator expressions that are returned within a `ciStructGeneratorExpressions` def struct. These expressions are evaluated by the `ciCreateModgen` SKILL function during modgen creation and modification. See also, ciSetStructGeneratorExpressions, ciGetStructGeneratorExpressions, and ciCreateModgen.

For a full description of the fields of the `ciStructGeneratorExpressions` def struct, see ciSetStructGeneratorExpressions.

Arguments

None

Value Returned

l_structureExpressions	A list of lists where each sub-list contains the structure type and its `ciStructureGeneratorExpression` def struct.

Example

```
ciListStructGeneratorExpressions()
    ((CascodedCurrentMirror ciStructGeneratorExpressions@0x224c3518)
    (CurrentMirror ciStructGeneratorExpressions@0x224c3500)
    (DiffPair ciStructGeneratorExpressions@0x224c34e8)
    (LargeMfactor ciStructGeneratorExpressions@0x224c34b8)
    (PassiveDeviceArray ciStructGeneratorExpressions@0x224c34d0)
)
```

ciListStructPDKMults

```
ciListStructPDKMults(
     )
     => g_pdkMults
```

Description

This function prints a list of all the registered structure PDK multipliers and returns a table of the registered structure type/PDK combinations. See also, ciSetStructPDKMult and ciGetStructPDKMult.

Arguments

None

Value Returned

g_pdkMults	A table of PDK multipliers for each registered structure type/PDK combination.

Example

```
pdkMultTable = ciListStructPDKMults()
            DiffPair.gpdk045 : 1.25
            DiffPair.gpdk090 : 1.25
        CurrentMirror.gpdk090 : 1.0
        CurrentMirror.gpdk045 : 1.0
CascodedCurrentMirror.gpdk045 : 1.0
CascodedCurrentMirror.gpdk090 : 1.0
         LargeMfactor.gpdk090 : 1.0
         LargeMfactor.gpdk045 : 1.0
                     Default. : 1.0
```

ciListStructTypes

```
ciListStructTypes(
     [ ?exclude l_exclude ]
     )
     => l_structSymbols
```

Description

Returns a list of the registered structure types used by the Circuit Prospector. See also, ciSetStructArgs, ciGetStructArgs, ciReplaceStructArg, ciSetStructPDKMult ciAddStructArg, ciGetStructArg, ciReplaceStructArg, ciDeleteStructArg, and ciRegexReplaceStructArgs.

Arguments

`?exclude` *l_exclude*	List of structure types to be excluded from the list.

Value Returned

l_structSymbols	List of structure type symbols.

Examples

```
ciListStructTypes()
=>
(Abutment CascodedCurrentMirror CurrentMirror DiffPair Dummies GuardRing
LargeMfactor NegativeSupply OrientationVertical PassiveDeviceArray PositiveSupply
Supply)
```

```
ciListStructTypes(?exclude '(Abutment Dummies GuardRing Supply))
=>
(CascodedCurrentMirror CurrentMirror DiffPair LargeMfactor NegativeSupply
OrientationVertical PassiveDeviceArray PositiveSupply)
```

ciListTemplateTypes

```
ciListTemplateTypes(
      )
      => l_templateTypeList
```

Description

Generates a list of the current constraint template types dictionary, outputs its details.

For each constraint template type is listed:

- The name of the type
- A list of the legal template constraint parameters, which contains:
  - The parameter name
  - Its data type name
  - A default value
  - A range of allowed values that constraint the parameter's range of values for a given data type.
  - SKILL Expressions (optional), and their current evaluation result: In the case of a parameter defined by SKILL expressions, a "dynamic parameter", its default value, or range of allowed values are defined dynamically, by SKILL expressions. These expressions can be currently evaluated or not yet evaluated.

The keyword `evaluated` or `nonevaluated` indicates whether the expressions are currently evaluated or at the opposite are not yet evaluated.

Note: The creation of the first template instance of a given template type will trigger the evaluation of the parameters' expressions for the current template type, in the context of the current virtuoso session.

Note: For more details on template parameters and dynamic parameters defined by SKILL expressions, refer to the Template Parameters section in the *Virtuoso Unified Custom Constraints User Guide*.

Arguments

None

Value Returned

l_templateTypeList

A list of the template types of the following form:

```
("<Constraint_Template_Type_Name>"
    (("<Parameter1 Name>" <dataTypeName>
<defaultValue> [value range] [enum choices
definition]
        <evaluated or unevaluated>
        )
    ("<Parameter2 with expression Name>"
<dataTypeName> <defaultValueExpression>
[valueRangeExpression]
[enumChoiceExpression]
        <evaluated or unevaluated>
        )
    )

    ...

    "<ParameterN Name>" <dataTypeName>
<defaultValue> [value range] [enum choices
definition]
        <evaluated or unevaluated>
        )
)
```

Example

Prerequisite: Open a schematic with VSE XL or layout with Layout XL.

```
ciListTemplateTypes()
(("CurrentMirror" ;; template type name
    (("Style" enum nil enum "\"Auto SingleRow DoubleRow\"" ;;example of enum
    parameter with an enum choice definition.
        unevaluated
        )
        ("Add Dummies" boolean nil boolean "nil"
        unevaluated
        )
        ("Abut All" boolean nil boolean "t" ;; example of boolean type parameter,
        with a nil default value
        unevaluated
        )
        ("Add GuardRing" boolean nil boolean "nil"
```

```
        unevaluated
        )
        ("Type" enum nil enum "\"ring pane\""
        unevaluated
        )
        ("Shape" enum nil enum "\"rectangular rectilinear\""
        unevaluated
        )
        ("Spacing" float nil float "0.0"
        unevaluated
        )
        ("Left Spacing" float nil float "0.0"
        unevaluated
        )
        ("Right Spacing" float nil float "0.0"
        unevaluated
        )
        ("Top Spacing" float nil float "0.0"
        unevaluated
        )
        ("Bottom Spacing" float nil float "0.0"
        unevaluated
        )
        ("MPP" enum nil enum "strcat(buildString(ciGetTechMPPNames()))" ;; example
        of enum parameter whose enum choices are defined by a SKILL expression.
        unevaluated ;; this parameter's expressions are currently unevaluated.
        )
        ("Use Fluid" boolean nil boolean "nil"
        unevaluated
        )
        ("Width" float nil float "0.0"
        unevaluated
        )
        ("Use Min DRC for Spacing" boolean nil boolean "nil"
        unevaluated
        )
    )
    )
("matchNewCB"
    (("strength" enum nil enum "\"low\""
        "list(\"low\" \"medium\" \"high\")" unevaluated
```

```
        )
        ("doubleNoDefault" float 0.0)
        ("doubleWDefault" float nil float "3.45"
        "2.4" unevaluated
        )
        ("doubleWDefault2" float nil float "1.4"
        "0.2" "4.4" unevaluated
        )
        ("intNoDefault" int 0)
        ("intWDefault" int nil int "4"
        "2" unevaluated
        )
        ("intWDefault2" int nil int "4"
        "2" "7" unevaluated
        )
    ) ))
```

ciMemberIndexToModgenPatternSymbol

```
ciMemberIndexToModgenPatternSymbol(
     x_memIndx
     )
     => t_patternSymbol
```

Description

This function returns the modgen pattern symbol for a member where member index is the count of the number of unique member names that have so far been processed in the list of members.

Arguments

x_memIndx	The member index value.

Value Returned

t_patternSymbol	The modgen pattern symbol for the given `memberIndx`.

Examples

```
ciMemberIndexToModgenPatternSymbol(0)
=> "A"
ciMemberIndexToModgenPatternSymbol(1)
=> B
ciMemberIndexToModgenPatternSymbol(2)
=> C
ciMemberIndexToModgenPatternSymbol(26)
=> A1
ciMemberIndexToModgenPatternSymbol(55)
=> D2
ciMemberIndexToModgenPatternSymbol(500)
=> G19
```

ciModgenDummyNetName

```
ciModgenDummyNetName(
     l_devInfo
     )
     => t_dummyNetName
```

Description

Determines the modgen dummy net name based on bulk connectivity of the specified devices or the power/ground nets in the design. See also, ciCreateModgenDummy.

Arguments

l_devInfo	The modgen device information as created by ciCollectDeviceInfo.

Value Returned

t_dummyNetName	The dummy net name.

Example

```
devInfo = ciCollectDeviceInfo(cache '(("MN0" inst) ("MN1" inst)) )
dummyNetName = ciModgenDummyNetName(devInfo)
"VDD"
```

ciNumDevices

```
ciNumDevices(
     g_cache
     l_devPath
     )
     => x_numDevs
```

Description

Returns the number of physical devices for a given device name in the schematic, which effectively retrieves the mfactor for the device.

Arguments

g_cache	The constraints cache.
l_devInfo	A list containing the full path to the device and the member type.

Value Returned

x_numDevs	The number of physical devices.

Example

```
ciNumDevices(cache list("/I1/MN1" 'inst))
=> 4
```

ciPadModgenPattern

```
ciPadModgenPattern(
     l_pattern
     x_rowCount
     x_colCount
     [ ?addDummies l_addDummies ]
     )
     => l_pattern
```

Description

Adds any missing elements in the modgen pattern list for the specified row and column counts.

Arguments

l_pattern	The modgen pattern which is a list containing two sub-lists. The first sub-list represents the device pattern in each row and the second sub-list represents the device orientations in each row.
x_rowCount	The number of modgen rows.
x_colCount	The number of modgen columns.
?addDummies *l_addDummies*	Whether dummies should be added for any missing elements.

Value Returned

l_pattern	The added modgen pattern.

Examples

```
ciPadModgenPattern( list(list( '( A  B  )  '( C  D  ) ) list( '( R0  R0  ) '( MX
MX  ))) 3 3)
=>(((A B \-) (C D \-) (\- \- \-)) ((R0 R0 \-) (MX MX \-) (\- \- \-)))
```

Modgen has `3` rows and `3` columns but the pattern only specifies `2` rows and `2` columns.

ciRegexReplaceStructArgs

```
ciRegexReplaceStructArgs(
     l_args
     t_regularExpr
     t_toStr
     )
     => l_newArgs
```

Description

For the passed constraint generator argument list replace any strings matching the passed regular expression with the passed string replacement. This function is useful for customizing the default constraint generator arguments registered for a given structure. See also, ciGetStructArgs, ciSetStructArgs, ciReplaceStructArg, ciAddStructArg, ciDeleteStructArg, and ciGetStructArg.

Arguments

l_args	The constraint generator arguments to be updated.
t_regularExpr	A regular expression to be matched within the passed constraint generator args.
t_toStr	What the matching string should be replaced with.

Value Returned

l_newArgs	The updated constraint generator arguments (the original args are unchanged).

Example

To update the current mirror structure args to use a current mirror specific PDK multiplier instead of the default:

```
ciRegexReplaceStructArgs(currentMirrorArgs "ciGetStructPDKMult('Default)"
"ciGetStructPDKMult('CurrentMirror)")
```

ciReinitStructTemplateDefs

```
ciReinitStructTemplateDefs(
     )
     => l_templateStructTypesInitialized
```

Description

Re-initializes the template definitions created by the Circuit Prospector for registered structures. This is necessary if any of the structure arguments are modified. Typically, these modifications will be in the libInit.il file loaded when the PDK library is loaded or in a .cdsinit file.

See also, ciListStructTypes, ciSetStructArgs, ciGetStructArgs, ciReplaceStructArg, ciSetStructPDKMult, ciAddStructArg, ciGetStructArg, ciReplaceStructArg, ciDeleteStructArg, and ciRegexReplaceStructArgs.

Arguments

None

Value Returned

l_templateStructTypesInitialized	A list of the structure type symbols that have had their template definitions re-initialized.

Example

```
ciReinitStructTemplateDefs()
=>
(CascodedCurrentMirror CurrentMirror DiffPair LargeMfactor PassiveDeviceArray)
```

ciRemoveSymmetricPinAlignments

```
ciRemoveSymmetricPinAlignments(
     g_cache
     )
     => t / nil
```

Description

This utility function used by the Rapid Analog Prototype Enforce Precedence constraint generator to remove alignment constraints on symmetric pins.

Arguments

g_cache	The constraints cache for which alignment constraints are to be removed.

Value Returned

`t`	Alignment constraints removed successfully.
`nil`	Command failed.

Example

```
mapcar(lambda((con) list(con->type mapcar(lambda((mem) car(mem)) con->members) ) )
cache->constraints)
((alignment ("vcom:5" "vcop:3"))
(symmetry  ("vcom:5" "vcop:3")))
```

```
ciRemoveSymmetricPinAlignments(cache)
```

```
mapcar(lambda((con) list(con->type mapcar(lambda((mem) car(mem)) con->members) ) )
cache->constraints)
((symmetry ("vcom:5" "vcop:3")))
```

ciReplaceStructArg

```
ciReplaceStructArg(
     s_structType
     t_argName
     l_newArg
     )
     => t / nil
```

Description

For the passed structure type this function replaces the named constraint generator argument with a new argument definition. See also, ciGetStructArgs, ciSetStructArgs, ciRegexReplaceStructArgs, ciAddStructArg, ciDeleteStructArg, and ciGetStructArg.

Arguments

s_structType	The structure type.
t_argName	The name of the constraint generator argument to be replaced.
l_newArg	The new argument definition, which is a list defining the new argument to be added in the form list (<t_argName> <s_argType> <g_argVal | t_argExpr> [s_`hide])

Value Returned

`t`	The argument was replaced sucessfully.
`nil`	Command failed.

Examples

```
ciReplaceStructArg('DiffPair "Add GuardRing"  list("GuardRing" `bool t)
```

Enables the addition of guard rings by default for `DiffPair` structures.

ciSaveConstraintGenerator

```
ciSaveConstraintGenerator(
     t_constraintGeneratorName
     )
     => t / nil
```

Description

Saves a constraint generator in the `.cadence/dfII/ci/generators` directory with the name given to it. The spaces in the constraint generator's name are replaced by underscores.

Arguments

t_constraintGeneratorName	The name of the constraint generator.

Value Returned

`t`	The constraint generator was successfully saved in the default directory.
`nil`	Invalid constraint generator name.

Examples

To save constraint generator named `Module (Current Mirror)`, use the following SKILL command:

```
ciSaveConstraintGenerator("Module (Current Mirror)")
```

This saves the given constraint generator at
`.cadence/dfII/ci/generators/Module_(Current_Mirror).il`

ciSetStructArgs

```
ciSetStructArgs(
     s_structType
     l_args
     )
     => t / nil
```

Description

Sets the constraint generator arguments for the passed structure type. See also, ciGetStructArgs, ciRegexReplaceStructArgs, ciAddStructArg, ciDeleteStructArg. ciReplaceStructArg, and ciGetStructArg.

Arguments

s_structType	The structure type symbol.
l_args	A list of lists defining the constraint generator arguments for the given structure where each sub-list is in the form list `(<t_argName> <s_argType> <g_argVal | t_argExpr> [s_`hide])`

Value Returned

`t`	The constraint generator argument for the passed structure type successfully set.
`nil`	Failed to set the constraint generator argument for the passed structure type.

Examples

The following examples illustrate the use of the `ciSetStructArgs` SKILL command:

Example 1

```
ciSetStructArgs('DiffPair
    append(
    append(
    append(
    append(
    list(list("Style" `enum "\"Auto row1 row2symmetric MultiRowMatched\""))
```

```
ciGetStructArgs('Dummies)) ciGetStructArgs('Abutment))
ciGetStructArgs('GuardRing)) "ciGetStructPDKMult('Default)"
"ciGetStructPDKMult('DiffPair)")
)
)
```

To simplify the definition of structures that require dummies, abutment, and guardRings, sets of arguments are pre-registered for each of these using the struct types, `Dummies`, `Abutment`, and `GuardRing`. For example,

```
ciSetStructArgs('CurrentMirror
    append(
    append(
    append(
    append(list(list("Style" `enum "\"Auto SingleRowDoubleRow"\"))
ciGetStructArgs(Dummies)) ciGetStructArgs(Abutment)) ciGetStructArgs(GuardRing))
)
)
```

Example 2

```
ciSetStructArgs('Cascode
    append(
        list( list("Style" `enum "\"Auto SingleRow DoubleRow\""))
        append( ciGetStructArgs('Dummies)
        append( ciGetStructArgs('Abutment)
        append( ciGetStructArgs('GuardRing)
    ciRegexReplaceStructArgs(ciGetStructArgs('"ciGetStructPDKMult
    ciGetStructPDKMult( Default)" "ciGetStructPDKMult('Cascode)"))))
    )
)
```

As shown above, this function registers the arguments for this structure type.

ciSetStructGeneratorExpressions

```
ciSetStructGeneratorExpressions(
     s_structType
     r_structExpressions
     )
     => t / nil
```

Description

Sets the constraint generator expressions for the passed structure type. The generator expressions are specified within a `ciStructGeneratorExpressions` def struct. These expressions are evaluated by the `ciCreateModgen` function during modgen creation and modification. See also, ciGetStructGeneratorExpressions, ciListStructGeneratorExpressions, and ciCreateModgen.

Arguments

s_structType	The structure type for which the generator expressions needs to be set.
r_structExpressions	A `ciStructGeneratorExpressions` def struct with each of its fields set to the generator expression string appropriate for the structure.

Value Returned

`t`	The constraint generator expressions for the passed structure type set successfully.
`nil`	Failed to set the constraint generator expressions for the passed structure type.

Example

```
ciSetStructGeneratorExpressions('DiffPair
    make_ciStructGeneratorExpressions(
        ?pattern   "ciGenerateDiffPairPattern(devInfo args)"
        ?guardRing "ciCreateGuardRing(cache modgen args)"
        ?routing   "args->Route"
        ?topology  "topology = createDiffPairTopology(cv modgen devInfo
        modgenPattern modgenPatternColumnised)"
```

```
        ?modgenPreCreateOrModify  "modgenParams = ciMergeParams(modgenParams
        list(list(\"mergeLayer\"  args->\"Merge Layer\")))"
        ?modgenPostCreateOrModify "reorderModgenMembersByDeviceMapping(modgen
        args->\"Device Mapping\")"
))
```

Here:

- `?pattern` is an expression for generating the interdigitation pattern from the passed devices. The expression should return a list of lists where each sub-list represents a row in the pattern. The sub-list should be a list of device names for that row and optionally device modgen parameters. For example, to specify a `"ABBA/BAAB"` pattern, this function will return:

  ```
  '(
          ("MN0" "MN1" "MN1" "MN0")
          ("MN1" "MN0" "MN0" "MN1")
  )
  ```

  The device name in the list can be a simple string as above or it can be a list in normal constraint member form that can contain member parameters, for example,

  ```
  '(
            (("MN0" 'inst (("horiCustomSpacing" 0.123))) "MN1" "MN1" "MN0")
            ("MN1" "MN0" "MN0" "MN1")
  )
  ```

  Dummies are represented by `"*"` and gaps are represented by `"-"`. For example,

  ```
  '(
          ("*" "MN0" "MN1" "-" "MN1" "MN0" "*")
          ("*" "MN1" "MN0" "-" "MN0" "MN1" "*")
  )
  ```

- `?guardRing` is an expression for creating a guard ring for the modgen using the specified arguments, for example, `ciCreateGuardRing(cache modgen args)`.
- `?routing` is an expression that should evaluate to `t` or `nil` to indicate whether modgen routing is required.
- `?topology` is an expression that generates the database topology objects for a modgen. This expression will be evaluated only if the `?routing` expression returns `t`. The expression should assign the topology to the topology variable.
- `?modgenPreCreateOrModify` is an expression to be evaluated before modgen creation or modification. This is used for updating the modgen parameters. For example, supplying additional parameters such as `mergeLayers`.

  Note: The `rows`, `columns`, `pattern`, and `routeStyle` parameters are automatically derived by `ciCreateModgen()`. The expression should reference and update `modgenParams` that is the parameters list in the usual constraint parameter form, that is, a list of sub-lists where each sub-list is of the form `(<paramName> <paramValue>)`.

- `?modgenPostCreateOrModify` is an expression to be evaluated after the modgen has been created or modified, but before it has been routed. This allows for any final modgen modifications before it is routed. For example, the *GenericModgen* generator reorders members to match the symbol mapping, or the user-defined function, `reorderModgenMembersByDevicMapping()`, in the example code above has been written for reordering the members to achieve the desired device mapping in the modgen.

  Note: The modgen constraint modifications after routing will invalidate the routing and the modifications might not be applied.

The `ciCreateModgen()` function will call expressions to create the modgen device pattern and guard ring. These expressions will be pre-registered for each of the structure types.

It is possible to register alternative modgen pattern, and guard ring expressions using `ciSetStructGeneratorExpressions()`.

A SKILL structure `ciStructGeneratorExpressions` is defined with pattern and guard ring fields. These fields will be SKILL expressions that will be called to generate the modgen device pattern, and guardring.

By default, the MOS Current Mirror generator expressions will be registered as mentioned below:

```
ciSetStructGeneratorExpressions('CurrentMirror
    make_ciStructGeneratorExpressions(
        ?pattern "ciGenerateCurrentMirrorPattern(devInfo args)"
        ?guardRing "when(args->\"Add GuardRing\" ciCreateGuardRing(cache modgen
        args))"
    )
)
```

Where

- `ciGenerateCurrentMirrorPattern()` returns either single row or double row device pattern lists dependent on the settings in the generator arguments.
- `ciCreateGuardRing()` creates a guard ring for the modgen based on the guard ring settings in the passed generator arguments.

As shown above, the following function registers the generator expressions for this structure type:

```
ciSetStructGeneratorExpressions('Cascode
    make_ciStructGeneratorExpressions(
       ?pattern "generateCascodeModgenPattern(devInfo args)"
       ?guardRing "when(args->\"Add GuardRing\" createCascodeGuardRing(cache
        modgen args))"
    )
)
```

Pattern Field Expression

The pattern field expression of the `ciStructGeneratorExpressions` structure should evaluate to a list of lists where each sub-list represents a row in the modgen and contains device names and optionally, a list of modgen device parameters.

The format of the device names and parameters is similar to that is used when specifying the members when calling `ciConCreate()`, although the member type is not required.

The registered pattern expression can reference the cache, `devInfo`, and arguments as variables where, `devInfo` is the device information DPL returned by the new `ciCollectDeviceInfo()` function and arguments are disembodied property list containing the constraint generator arguments and values specified when the constraint generator was run.

For example, to represent the pattern `ABBA/BAAB` for a diff pair with mfactor `4`, with devices named `MN1` and `MN2`, and custom spacings for some of the devices in the pattern the pattern list would be:

```
'( ((“MN1” ((“horizontalCustomSpacing” 0.2))) (“MN2” ((“horizontalCustomSpacing”
0.3))) (“MN2”) (“MN1”))
((“MN2” ((“horizontalCustomSpacing” 0.2))) (“MN1” ((“horizontalCustomSpacing”
0.3))) (“MN1”) (“MN2”)) )
```

The `ciCreateModgen()` function takes this information and assign the correct row/column numbers and modgen pattern string for the modgen being created.

The `devs` field of the `devInfo` variable is a list of disembodied property lists where each disembodied property list contains PDK independent information about a device in the structure. This information can be used to generate the required modgen pattern lists.

For example, to generate a single row modgen containing all devices in the structure you could write the following procedure:

```
procedure( createSingleRowPattern(cache devInfo args)
    let( (pattern row)
        row = list()
            foreach(dev devInfo->devs
            for(m 1 dev->mFactor
        row = cons(dev->name row )
        )
    )
```

```
    pattern = list(reverse(row))
    )
)
```

Then, set the pattern expression field of the `ciStructGeneratorExpressions` structure to call this function, as shown below:

```
genExprs = ciGetStructGeneratorExpressions('CurrentMirror)
genExprs->pattern = "createSingleRowPattern(cache devInfo args)"
ciSetStructGeneratorExpressions('CurrentMirror genExprs)
```

You can also use of the `devInfo` disembodied property list, as shown in the example below:

```
;;; ABBA
;;; BAAB
;;;
;;; Note args contains user defined properties spacing1 g p p p g and spacing2 for
specifying the device spacings
;;;
procedure( ciDiffPairPatternMFactor4( cache devInfo args)
    prog( (devA devB row0 row1 pattern)
        unless(length(devInfo->devs) == 2 warn("Wrong number of devices\n")
        return())
        devA = car(devInfo->devs)
        devB = car(devInfo->devs)
        unless(devA->mFactor == 4 && devB->mFactor == 4 warn("wrong mfactor")
        return())
        row0 = list( list(devA->name args->spacing1) list(devB->name args
        ->spacing2)
        list(devB->name) list(devA->name))
        row1 = list( list(devB->name args->spacing1) list(devA->name args
        ->spacing2)
        list(devA->name) list(devB->name))
        pattern = list(row0 row1)
        return(pattern)
    )
)
```

GuardRing Field Expression

The guardRing field of the `ciStructGeneratorExpressions` structure should be an expression that generates a guard ring for the passed modgen based on the guard ring settings in the passed generator arguments.

By default, the guardRing expression for all structure types will be registered as:

```
"when(args->\"Add GuardRing\" ciCreateGuardRing(cache modgen args))"
```

The example below shows how to register alternative guard ring generator expressions for structure types:

```
procedure( createGuardRing(cache modgen args)
    let( (guardRingNet guardRingMembers guardRingParams guardRing)
        when(args->"Add GuardRing"
```

```
            guardRingNet = list(args->Net 'net list( list("shape" args->Shape)
            list("spacing" args->Spacing) list("mppName" args->MPP)))
            guardRingMembers = list(list(modgen->name 'modgen) guardRingNet)
            guardRingParams = list( list( "type" args->Type ) )
            guardRing = ciConCreate(cache 'powerStructure ?members
            guardRingMembers ?params
            guardRingParams ?verbose nil)
            )
        )
)
genExprs = ciGetStructGeneratorExpressions('CurrentMirror)
genExprs->guardRing = "createGuardRing(cache modgen args)"
ciSetStructGeneratorExpressions('CurrentMirror genExprs)
```

ciSetStructPDKMult

```
ciSetStructPDKMult(
     s_structType
     t_pdkName
     f_pdkMultiplier

     )
     => t
```

Description

Sets the PDK specific multiplier for the passed structure type and PDK name. The PDK multipliers are typically used within constraint generator argument expressions to scale the values in a PDK independent way. See also, ciGetStructPDKMult.

Arguments

s_structType	The structure type for which the PDK multiplier is retrieved.
t_pdkName	The name of the PDK.
f_pdkMultiplier	The PDK specific multiplier.

Value Returned

t	Sets the PDK specific multiplier for the passed structure type and PDK name.

Examples

```
ciSetStructPDKMult('DiffPair "gpdk090" 1.15)
ciSetStructPDKMult('DiffPair "gpdk045" 1.75)
```

ciSortDeviceInfoByFingerWidth

```
ciSortDeviceInfoByFingerWidth(
     l_devInfo
     )
     => l_sortedDevInfo
```

Description

Returns devs ordered by finger width. The device with the largest finger width will appear first in the list.

A prerequisite for this to work is that the device finger width parameter name should have been registered using ciMapParam. To check the parameter has been registered type ciPrintMappedParams and look at the parameter names registered for `fingerWidth`.

Arguments

l_devInfo	Specifies the constraint generator device information disembodied property list as returned by `ciCollectDeviceInfo`.

Value Returned

l_sortedDevInfo	Sorted device info where the device with the largest finger width will appear first in the list.

Example

```
devInfo = ciCollectDeviceInfo(cache '(("MN1" inst) ("MN2" inst) ("MN3" inst)))-
>devs
mapcar(lambda((dev) list(dev->name dev->fingerWidth)) devInfo)
    (("MN1" 1.1e-05) ("MN2" 1.8e-05) ("MN3" 1.4e-05))
sortedDevInfo = ciSortDeviceInfoByFingerWidth(devInfo)
mapcar(lambda((dev) list(dev->name dev->mFactor)) sortedDevInfo)
    (("MN2" 1.8e-05) ("MN3" 1.4e-05) ("MN1" 1.1e-05))
```

ciSortDeviceInfoByMfactor

```
ciSortDeviceInfoByMfactor(
     l_devInfo
     )
     => l_sortedDevInfo
```

Description

Returns devices ordered by mfactor. The device with the largest mfactor will appear first in the list.

A prerequisite for this to work is that the device mfactor parameter name should have been registered using ciMapParam. To check the parameter has been registered ciPrintMappedParams and look at the parameter names registered for `mFactor`.

Arguments

l_devInfo	Specifies the constraint generator device information disembodied property list as returned by `ciCollectDeviceInfo`.

Value Returned

l_sortedDevInfo	The devices ordered by mfactor. The device with the largest mfactor will appear first in the list.

Example

```
devInfo = ciCollectDeviceInfo(cache '(("MN1" inst) ("MN2" inst) ("MN3" inst)))->
devs
mapcar(lambda((dev) list(dev->name dev->mFactor)) devInfo)
    (("MN1" 1) ("MN2" 8) ("MN3" 4))
sortedDevInfo = ciSortDeviceInfoByMfactor(devInfo)
mapcar(lambda((dev) list(dev->name dev->mFactor)) sortedDevInfo)
    (("MN2" 8) ("MN3" 4) ("MN1" 1))
```

ciSortDeviceInfoByX

```
ciSortDeviceInfoByX(
     l_devInfo
     [ ?margin f_margin ]
     )
     => l_sortedDevInfoDevs
```

Description

Returns devices ordered by increasing `X`. Margin defines the margin for equivalent `X` coordinates.

Arguments

l_devInfo	Specifies the constraint generator device information disembodied property list as returned by `ciCollectDeviceInfo`.
`?margin` *f_margin*	Defines the margin for equivalent `X` coordinates.

Value Returned

l_sortedDevInfoDevs	Returns the devInfo devices ordered by increasing `X`.

Example

```
devInfo = ciCollectDeviceInfo(cache '(("MN1" inst) ("MN2" inst) ("MN3" inst)))-
>devs
mapcar(lambda((dev) list(dev->name car(dev->dbId->xy))) devInfo)
    (("MN3" 1.4) ("MN2" 1.8) ("MN1" 1.1))
sortedDevInfo = ciSortDeviceInfoByX(devInfo)
mapcar(lambda((dev) list(dev->name car(dev->dbId->xy))) sortedDevInfo)
    (("MN1" 1.1) ("MN3" 1.4) ("MN2" 1.8))
```

ciSortDeviceInfoByXY

```
ciSortDeviceInfoByXY(
     l_devInfo
     [ ?margin f_margin ]
     )
     => l_sortedDevInfoDevs
```

Description

Returns devices ordered by increasing `X` and increasing `Y` where, devices have same `X`. Margin defines the margin for equivalent `X/Y` coordinates.

Arguments

l_devInfo	Specifies the constraint generator device information disembodied property list as returned by `ciCollectDeviceInfo`.
?margin *f_margin*	Defines the margin for equivalent `X/Y` coordinates.

Value Returned

l_sortedDevInfoDevs	Devices ordered by increasing `X` and increasing `Y` where, devices have same `X`.

Example

```
devInfo = ciCollectDeviceInfo(cache '(("MN1" inst) ("MN2" inst) ("MN3" inst)))-
>devs
mapcar(lambda((dev) list(dev->name dev->dbId->xy)) devInfo)
    (("MN1" (-0.1875 1.1875)) ("MN2" (-0.1875 1.8125)) ("MN3" (-0.875 1.8125)))
sortedDevInfo = ciSortDeviceInfoByXY(devInfo)
mapcar(lambda((dev) list(dev->name dev->dbId->xy)) sortedDevInfo)
    (("MN3" (-0.875 1.8125)) ("MN1" (-0.1875 1.1875)) ("MN2" (-0.1875 1.8125)))
```

ciSortDeviceInfoByY

```
ciSortDeviceInfoByY(
     l_devInfo
     [ ?margin f_margin ]
     )
     => l_sortedDevInfoDevs
```

Description

Returns devices ordered by increasing `Y`. Margin defines the margin for equivalent `Y` coordinates.

Arguments

l_devInfo	Specifies the constraint generator device information disembodied property list as returned by `ciCollectDeviceInfo`.
?margin *f_margin*	Defines the margin for equivalent `Y` coordinates.

Value Returned

l_sortedDevInfoDevs	Returns the devices ordered by increasing `Y`.

Example

```
devInfo = ciCollectDeviceInfo(cache '(("MN1" inst) ("MN2" inst) ("MN3" inst)))-
>devs
mapcar(lambda((dev) list(dev->name cadr(dev->dbId->xy))) devInfo)
    (("MN1" 3.4) ("MN2" 3.8) ("MN3" 3.1))
sortedDevInfo = ciSortDeviceInfoByY(devInfo)
mapcar(lambda((dev) list(dev->name cadr(dev->dbId->xy))) sortedDevInfo)
    (("MN3" 3.1) ("MN1" 3.4) ("MN2" 3.8))
```

ciSortDeviceInfoByYX

```
ciSortDeviceInfoByYX(
     l_devInfo
     [ ?margin f_margin ]
     )
     => l_sortedDevInfoDevs
```

Description

Returns devices ordered by increasing `Y` and increasing `X` where devices have same `Y`. Margin defines the margin for equivalent `X/Y` coordinates.

Arguments

l_devInfo	Specifies the constraint generator device information disembodied property list as returned by `ciCollectDeviceInfo`.
`?margin` *f_margin*	Defines the margin for equivalent `X/Y` coordinates.

Value Returned

l_sortedDevInfoDevs	Devices ordered by increasing Y and increasing X where devices have same Y.

Example

```
devInfo = ciCollectDeviceInfo(cache '(("MN1" inst) ("MN2" inst) ("MN3" inst)))-
>devs
mapcar(lambda((dev) list(dev->name dev->dbId->xy)) devInfo)
    (("MN1" (-0.1875 1.1875)) ("MN2" (-0.1875 1.8125)) ("MN3" (-0.875 1.8125)))
sortedDevInfo = ciSortDeviceInfoByYX(devInfo)
mapcar(lambda((dev) list(dev->name dev->dbId->xy)) sortedDevInfo)
    (("MN1" (-0.1875 1.1875)) ("MN3" (-0.875 1.8125)) ("MN2" (-0.1875 1.8125)))
```

ciUnexpandPhysicalDeviceInfo

```
ciUnexpandPhysicalDeviceInfo(
     l_devInfo
     [ ?unexpandIterated g_unexpandIterated ]
     )
     => l_devInfo
```

Description

This is a utility function used in modgen generation for converting a physical `devInfo` disembodied property list into a logical devInfo disembodied property list. The `devInfo` disembodied property lists are created by calling `ciCollectDeviceInfo` for schematic (logical) and layout (physical) devices.

Modgens in layout have `mFactor` expanded physical names, such as `|NM1.1 |M1.2 |NM1.3` each with an mFactor of `1`. These names need to be unexpanded in the device info, in this case to `|NM1` with an mFactor of `3`. The function also handles the case where only a subset of the mfactored devices appear in the modgen, such as `|NM1.1 |NM1.3`. In this case, the mFactor is set to `2`.

Arguments

l_devInfo — Specifies the physical device disembodied property list created by calling `ciCollectDeviceInfo` in layout.

?unexpandIterated *g_unexpandIterated*

Controls whether individual iterated devices should be unexpanded back to their iterated device names. For example, `"MN1<1>" "MN1<2>"` becomes `"MN1<1:2>"`. Default: `nil`

Values Returned

l_devInfo — Returns the unexpanded device disembodied property list.

Example

```
firstPhysDev = car(physDevInfo->devs)
firstPhysDev->name
"|NM1.1"
firstPhysDev->mFactor 1
logDevInfo = ciUnexpandPhysicalDeviceInfo(physDevInfo)
firstLogDev = car(logDevInfo->devs)
```

```
firstLogDev->name "|NM1"
firstPhysDev->mFactor 3
```

ciUpdateModgenParamsAndMembers

```
ciUpdateModgenParamsAndMembers(
     g_modgen
     l_newModgenParams
     l_newModgenMembers
     )
     => t / nil
```

Description

A utility function, which updates the passed modgen constraint parameters and members.

Arguments

g_modgen	The modgen constraint for which the constraint parameters and members need to be updated.
g_newModgenParams	A list of the modgen parameters to be updated.
g_newModgenMembers	A list of the new members for the modgen.

Value Returned

`t`	The passed modgen constraint parameters and members were updated successfully.
`nil`	Failed to udpate the passed modgen constraint parameters and members.

Example

```
paramDPL = ciConvertParamsToDPL(modgen->parameters)
(nil numRows 1 numCols 4 addDummyRowCol nil pattern "interdigit 1" mergeLayer
"default"
paramDPL->numRows = 2
paramDPL->numCols = 2
newParams = ciConvertParamsDPLToParams(paramDPL)
mapcar(lambda((mem) car(mem)) modgen->members)
("MP5" "MP6" "MP9" "MP10")
newMembers = '(("MP10" inst) ("MP9" inst) ("MP6" inst) ("MP5" inst))
ciUpdateModgenParamsAndMembers(modgen newParams newMembers)
newParamDPL = ciConvertParamsToDPL(modgen->parameters)
(nil numRows 2 numCols 2 addDummyRowCol nil pattern
"interdigit 1" mergeLayer "default")
mapcar(lambda((mem) car(mem)) modgen->members)
("MP10" "MP9" "MP6" "MP5")
```

ciUtilsGetArgVal

```
ciUtilsGetArgVal(
     l_args
     t_argName
     t_argAltName
     [ ?defVal g_defVal ]
     )
     => g_val
```

Description

This is a utility function for accessing constraint generator argument values from a disembodied property list where the argument names may take one of two forms.

Arguments

l_args	A disembodied property list containing argument names and values.
t_argName	The name of the argument to find the value for.
t_argAltName	An alternative name for the argument to find the value for.
?defVal *g_defVal*	The default value to use if an argument with `argName` or `argAltName` cannot be found.

Value Returned

g_val	The value associated with `argName`, `altArgName`, or the `defVal` if none of the argument names can be found.

Examples

```
ciUtilsGetArgVal(args "Source Layer" "sourceLayer")
=> "Metal1"
ciUtilsGetArgVal(args "Gate Width"    "gateWidth")
=> 0.1
ciUtilsGetArgVal(args "DrainNNNN Width"    "drainNNNNNWidth" ?defVal 0.123)
=> 0.123
```

ciUtilsMakeNumberRange

```
ciUtilsMakeNumberRange(
     x_from
     x_to
     [ ?descending g_descending ]
     [ ?fmt t_fmt ]
     )
     => l_result
```

Description

This function returns a list containing a sequence of numbers ranging from the passed from value to the passed to value. Optionally, this function reverses the list into descending order and applies a `sprintf` format to the numbers.

Arguments

x_from	Start of the number range.
x_to	End of the number range.
?descending *g_descending*	Boolean to return the number range in descending order.
?fmt *t_fmt*	Apply the format string to each number in the range.

Value Returned

l_result	A list containing the number sequence in the specified range.

Examples

```
ciUtilsMakeNumberRange(1 10)
=> (1 2 3 4 5 6 7 8 9 10)
ciUtilsMakeNumberRange(1 5 ?descending t ?fmt "row%d" )
=> ("row5" "row4" "row3" "row2" "row1")
```

Custom Constraints Functions: Examples

This section contains SKILL function usage examples detailing how to:

- Print all Constraints on a Given Object (Sorted by Name)
- Print Context Status on a Given Object
- Print All Constraints (Sorted by Name) on all Objects (Sorted by Name)
- Print In-Context Status for all Objects (Sorted by Name)
- Print all Constraints Content (In Constraint Name Order)
- Get an Axis Parameter
- Get Parameters for Duplicating Constraints with Another Set of Members
- Print a Report File with all Constraints in the Cache
- Helper Functions for Sorting

Print all Constraints on a Given Object (Sorted by Name)

```
defun( ciTestPrintObjectConstraintsSorted (cache obj)
    printf("Object %L has constraints %L.\n" obj
        mapcar('ciConGetName
        sort( apply( 'ciObjectListCon
        append( (list cache) obj))
                       'ciTestConstraintLessp))))
```

Print Context Status on a Given Object

```
defun( ciTestPrintObjectContextStatus (cache obj)
    let( ((incxt apply( 'ciObjectIsInContext append( (list cache) o))) frase( "out-
    of-context"))
    when( incxt frase = "in-context")
        printf( "Object %L is %s.\n" obj frase)
           ))
```

Print All Constraints (Sorted by Name) on all Objects (Sorted by Name)

```
;; returns a list of objects sorted by name
    defun( ciTestPrintAllObjectsAllCon (cache)
      objects = sort( (ciCacheListConstrainedObjects cache) 'ciTestAssocListLessP)
        foreach( o objects apply( 'ciTestPrintObjectConstraintsSorted
        append( (list cache) (list o)))))
```

Print In-Context Status for all Objects (Sorted by Name)

```
;; returns a list of objects sorted by name
    defun( ciTestPrintAllObjectsContextStatus (cache)
      objects = sort( (ciCacheListConstrainedObjects cache) 'ciTestAssocListLessP)
        foreach( o objects apply( 'ciTestPrintObjectContextStatus
        append( (list cache) (list o))))
           t
           )
```

Print all Constraints Content (In Constraint Name Order)

```
;; returns t
    defun( ciTestPrintAllConAllContent (cache)
           objects = sort( (ciCacheListCon cache) ’ciTestConstraintLessp)
        foreach( o objects (ciTestPrintConstraintContent o))
           t)

defun( ciTestPrintConstraintContent (con)
    printf( "\nConstraint named %s has:\n" (ciConGetName con))
    printf( " Axis: %L\n" (ciConGetAxisName con))
    printf( " Members (always ordered as set):\n")
    pprint( (ciConListMembers con))
    printf( "\n Parameters:\n")
    pprint( (ciConListParams con))
       t)
```

Get an Axis Parameter

```
defun( ciTestGetCiAxisCreateParams cache( axisName)
   ;; this returns a list of parameters that, if passed to ciAxisCreate,
   ;; would create a duplicate of the axis passed in
   ;; just add the cache in front of each of them
    list( axisName (ciAxisListParams cache axisName))
     )
```

Get Parameters for Duplicating Constraints with Another Set of Members

```
defun( ciTestGetCiConCreateParams (con)
   ;; this returns a list of parameters that, if passed to ciConCreate,
   ;; would create a cuplicate of the constraint passed in
   ;; just add the cache in front of each of them
    let( (ret)
    when( con
       ret = list( (ciConGetType con)
                   ?name (ciConGetName con)
                   ?members (ciConListMembers con)
                   ?params (ciConListParams con))
       )
     ret
     ))
```

Print a Report File with all Constraints in the Cache

```
defun( ciTestPrintReport (cache @key (fileName nil) (format 'skill))
    let( (file)
    when( fileName
       file = (outfile fileName "w")
    unless( file (warn "Cannot open file %L for writing" fileName))
       )

    fprintf( file ";; ===================================================\n" )
    fprintf( file ";; ===================================================\n" )
    fprintf( file ";; ===================================================\n" )
    fprintf( file ";; ===================================================\n" )
    fprintf( file ";; all axes from this cache\n" )
    fprintf( file "\n")
    fprintf( file "ciAxes = list( \n")

foreach( axis sort( (ciCacheListAxesNames cache) 'alphalessp)
    pprint( (ciTestGetCiAxisCreateParams cache axis) file)
    fprintf( file "\n")
                  )
    fprintf( file ") \n")
    fprintf( file ";; finished printing all Axes \n")
    fprintf( file ";; ===================================================\n" )
    fprintf( file ";; ===================================================\n" )
    fprintf( file "\n" )
    fprintf( file "\n" )

    fprintf( file ";; ===================================================\n" )
    fprintf( file ";; ===================================================\n" )
    fprintf( file ";; all constraints from this cache\n" )
    fprintf( file "\n")
    fprintf( file "ciConstraints = (list \n")

     objects = sort( (ciCacheListCon cache) 'ciTestConstraintLessp)
        foreach( o objects
              ;; fprintf( file "%L\n" (ciTestGetCiConCreateParams o)))
        pprint( (ciTestGetCiConCreateParams o) file)
        fprintf( file "\n")
              )
```

```
    fprintf( file ") \n")
    fprintf( file ";; finished printing all constraints \n")
    fprintf( file ";; ===================================================\n" )
    fprintf( file ";; ===================================================\n" )
    fprintf( file ";; ===================================================\n" )
    fprintf( file ";; ===================================================\n" )
             t
           ))
```

Helper Functions for Sorting

```
;; compare an association list (that is a list containing list( list(
 ;; "name1" ...) list( "name2" ...) ...)
    defun( ciTestAssocListLessP (x y)
    alphalessp( (car x) (car y)))

    defun( ciTestConstraintLessp (x y)
    alphalessp( (ciConGetName x) (ciConGetName y)))
```

2

CST Access SKILL Functions

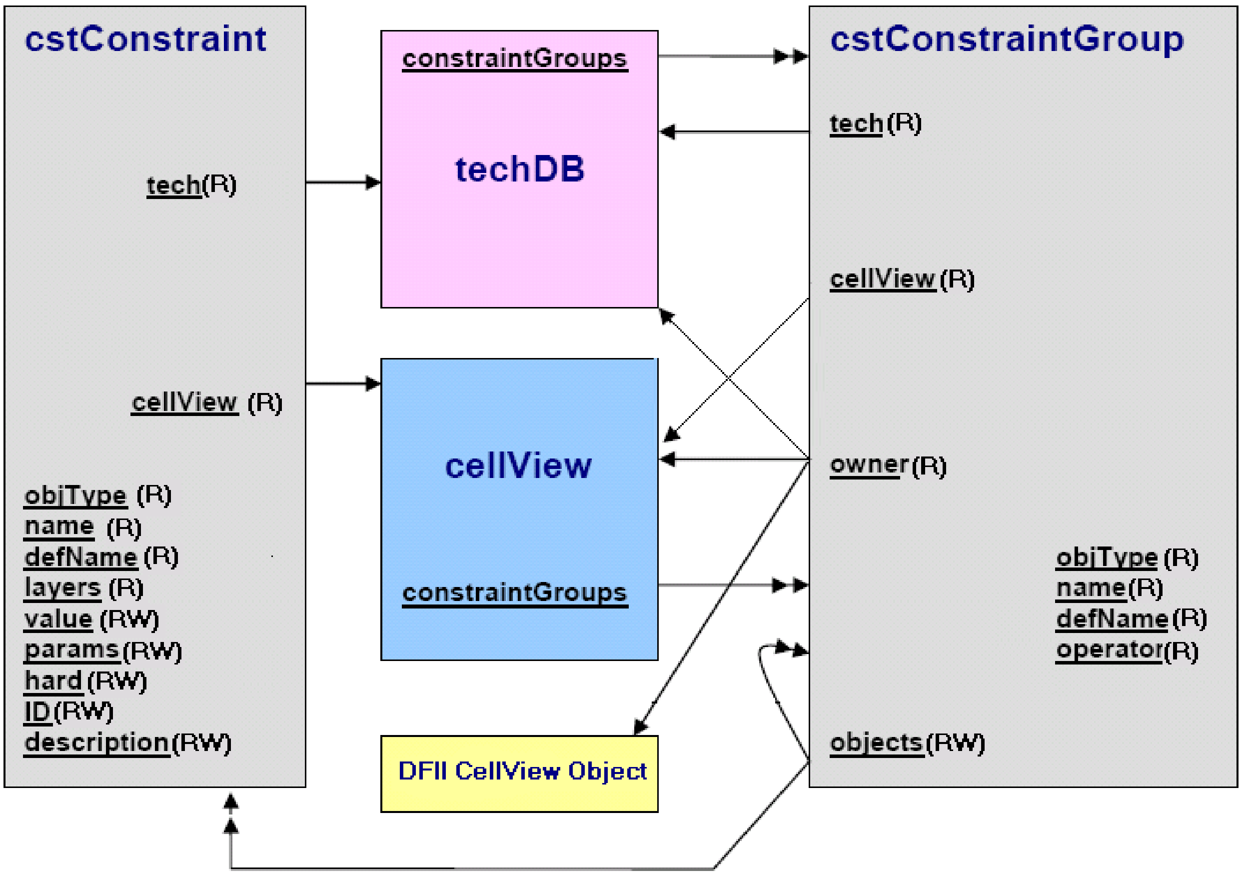
Introduction

This chapter provides information about DFII constraint objects using Cadence® SKILL language. There are two objects:

- **cstConstraintID** can be layer constraint, layer-pair constraint, layer array constraint, or a simple constraint. It contains attribute values and parameters. The attribute values are represented in integer, float, string, or SKILL list. The parameters are represented in a SKILL list.
- **cstConstraintGroupID** is the container of member constraints and member constraint groups.

The following figure displays the relationship between `cstConstraintID`, `cstConstraintGroupID`, `tech`, `cellview`, and `objects`, and their attributes. The attributes of `cstConstraintID` and `cstConstraintGroupID` have either *Read* (R) or *Read/Write* (RW) access.

Note: The owner attribute in the following figure is for a constraint group associated with a particular owning object. This attribute returns a pointer to the owning object.



The following example illustrates that you can get the attributes of a member `constraintGroup` and a `layerConstraint` from the `parentConstraintGroup`, which contains them:

```
parentConstraintGroup~>objects~>

returns
(
    (cst:0x080ef495 tech db:0x080ef892 cellview nil
     objType "constraintGroup" name "maximumYield" defName
     "userDefined" operator precedence owner nil
     objects (cst:0x080ee9ea cst:0x080ee9eb)
     )
    (cst:0x080ee9df tech db:0x080ef892 cellview nil
     objType "layerConstraint" name "C__187" defName
     "minSpacing" layers ("METAL2") value 0.555
     params nil hard nil ID
     nil description nil
    )
)
```

Constraint Functions

All constraint objects are defined by the constraint definition. Constraints always have a value associated with them and also a list of constraint parameters. There are four types of constraints: layer constraint, layer-pair constraint, layer array constraint, and simple constraint (has no layer).

The list of constraint functions is given below:

- cstCreateConstraint
- cstDeleteConstraint
- cstFindCutClassConstraintByName
- cstFindCutClassConstraintBySize
- cstGetUnreferencedConstraints
- cstGet1DTableValue
- cstGet2DTableValue
- cstGetTwoWidthTableValue
- cstIsId

cstCreateConstraint

```
cstCreateConstraint(
     g_cstConstraintGroupID
     t_constraintDefName
     g_layers
     g_value
     [ l_params ]
     [ g_isHard ]
     [ g_append ]
     [ t_name ]
     )
     => g_cstConstraintID / nil
```

Description

Creates a constraint in the constraint group (`g_cstConstraintGroupID`) with the specified constraint definition name (`t_constraintDefName`), layers, constraint value, constraint parameters, hard attribute, the position of the constraint with the members of the constraint group and the name of the constraint.

Arguments

g_cstConstraintGroupID	The ID of the constraint group, which is to own this constraint.
t_constraintDefName	Mapped DFII constraint name will be used to get a corresponding constraint definition name.
g_layers	List of layer names, layer numbers, or LPPs.
g_value	SKILL expression for the constraint value. It can be an integer, a float, or a list.

l_params	A SKILL list of constraint parameters. The default value is `nil`. The input constraint parameters are in the following format:
	`list(list('paramKeyWord{int | float | string | list})...)`
	For example,
	`cst1~>params = list(list('ConnectivityType "sameNet") ('list('PGNet t) =>((connectivityType "sameNet") (PGNet t))`

g_isHard	A Boolean indicating whether the constraint must be met. If the value is `nil`, it means that it is a soft constraint. By default, the value is `t`.
g_append	A Boolean having a value as either `nil` or `t`. Having the value as `nil` puts this constraint to the first member in the constraint group. The present first constraint group member is moved to the second and subsequent members are also moved down in order. However, if the value is `t`, then the constraint is put to the end of the member in the constraint group. By default, the value is `nil`.
t_name	This is a user defined constraint name. The name will be automatically created if it is not specified. The default value is `nil`.

Value Returned

g_cstConstraintID	Returns the ID of the constraint, which is created in the constraint group.
`nil`	Returns `nil` if the constraint is not created.

Example

- Layer constraint

```
tech     = techOpenTechFile("tech" "tech.db" "a")
foundryCG = cstFindConstraintGroupIn(tech "foundry" 't)
cst1      = cstCreateConstraint(foundryCG "viaSpacing" list("viaLayer") 0.4
              list(list('distance 0.5) list('numCuts 2)))
```

- Layer-pair constraint

```
cst2 = cstCreateConstraint(foundryCG "minSpacing" list("metal1" "metal2")1.2)
```

- Layer-array constraint

```
cst3 = cstCreateConstraint(foundryCG "minTouchingDirEnclosure"
                        list("metal1""metal2" "cont") 1.0)
```

- You can use `nil` for layers to create a simple constraint:

```
cst4 = cstCreateConstraint(foundryCG "validLayers" nil list("metal1""metal2"))
```

cstDeleteConstraint

```
cstDeleteConstraint(
     g_cstConstraintID
     )
     => t / nil
```

Description

Deletes the constraint from any of the constraint groups it is a part of. Its value and constraint parameters are also deleted.

Arguments

g_cstConstraintID	The identification of the constraint.

Value Returned

`t`	Returns `t` when `cstDeleteConstraint` successfully deletes the constraint.
`nil`	Returns `nil` when the pass-in constraint cannot be deleted.

Example

```
cstDeleteConstraint(cstnet1l)
```

cstFindCutClassConstraintByName

```
cstFindCutClassConstraintByName(
     d_constraintGroupId
     tx_layer
     t_name
     )
     => d_cutClassConstraintId / nil
```

Description

Returns the database ID of the cut class constraint with a given cut class name, layer, and constraint group.

For more details, see `cutClasses`.

Arguments

d_constraintGroupId	Database ID of the constraint group in which to search.
tx_layer	Layer number or layer name to be searched.
t_name	Name of the cut class.

Value Returned

d_cutClassConstraintId	Database ID of the cut class constraint.
`nil`	The cut class constraint was not found.

Example

Retrieves the database ID of a cut class constraint named `BAR0.2` on layer `BAR0` in the constraint group referenced by `grp`.

```
cstId = cstFindCutClassConstraintByName(grp "BAR0" "BAR0.2")
```

cstFindCutClassConstraintBySize

```
cstFindCutClassConstraintBySize(
     d_constraintGroupId
     tx_layer
     n_width
     n_length
     g_checkOrientation
     )
     => d_cutClassConstraintId / nil
```

Description

Searches the specified layer in the specified constraint group and returns the first cut class constraint with the specified width and length. If *`g_checkOrientation`* is set to `t`, the orientation of the cut class will also be included as part of the search.

For more details, see `cutClasses`.

Arguments

`d_constraintGroupId`	Database ID of the constraint group in which to search.
`tx_layer`	Layer number or layer name to be searched.
`n_width`	Width of the cut shape to be searched.
`n_length`	Length of the cut shape to be searched.
`g_checkOrientation`	When set to `t`, searches cut classes for which the `fixedOrientation` attribute is set to `t`. If none are found, searches for cut classes that have `fixedOrientation` set to `nil`.

Value Returned

`d_cutClassConstraintId`	Database ID of the cut class constraint.
`nil`	No matching cut class constraint was found on the specified layer.

Example

Retrieves the database ID of a cut class constraint with width `0.2`, length `0.3`, and a fixed orientation on the layer `BAR0`.

```
cstId = cstFindCutClassConstraintBySize(grp "BAR0" 0.2 0.3 t)
```

cstGetUnreferencedConstraints

```
cstGetUnreferencedConstraints(
     { d_techID | d_cellViewID }
     )
     => l_unreferencedConstraints / nil
```

Description

Returns all the unreferrenced constraints not belonging to any constraint group in a database (a technology database or a cellview).

Note: A slow performance could be experienced if the databases contain multiple constraints or constraint groups, and use either the `cstGetUnreferencedConstraints` or `cstGetConstraintGroups` API.

Arguments

d_techID	The identification of the technology database.
d_CellViewID	The identification of the cellview.

Value Returned

l_unreferencedConstraints	Returns the identification of all the unreferenced constraints.
nil	Returns `nil` if there are no unreferenced constraints available for a constraint group in a database.

Example

```
list_constraints = cstGetUnreferencedConstraints(techDB1)
foreach(danglingConstraint cstGetUnreferencedConstraints(cellViewId_1)
        cstDeleteConstraint(danglingConstraint)
```

cstGet1DTableValue

```
cstGet1DTableValue(
     d_ConstraintID
     n_index1
     )
     => n_value
```

Description

Returns the value of the table for the given index. The index varies depending on the constraint. It can be a width or a length.

Arguments

d_ConstraintID	The ID of the constraint.
n_index1	The index can be a width or a length.

Value Returned

n_value	The value of the table at the given index.

Example

```
cstNDrule2_Int1DTbl~>value =
list(list(list("width") 0.4)
list(0.1 0.15 0.2 0.25 0.3 0.35 0.4 0.45)
)
((("width") 0.4)
0.1 0.15 0.2 0.25 0.3
0.35 0.4 0.45
)
cstGet1DTableValue(cstNDrule2_Int1DTbl 0.1)
```

cstGet2DTableValue

```
cstGet2DTableValue(
     d_ConstraintID
     n_index1
     n_index2
     )
     => n_value
```

Description

Returns the value of the table for the given indexes.

Arguments

d_ConstraintID	The ID of the constraint.
n_index1	The index can be a width or a length, which usually correspond to the row of the table.
n_index2	The index can be a width or a length, which usually correspond to the column of the table.

Value Returned

n_value	The value of the table at the given indexes.

Example

```
Int2DTbl~>value
((("width" "length") 0.33)
((0.0 0.1) 0.22
(0.0 0.3) 0.32
(0.2 0.1)
0.23
(0.2 0.3) 0.0
(0.4 0.1) 0.24
(0.4 0.3) 0.34
)
)
cstGet2DTableValue(Int2DTbl 0.0 0.1)
```

cstGetTwoWidthTableValue

```
cstGetTwoWidthTableValue(
     d_ConstraintID
     n_index1
     n_index2
     n_index3
     )
     => n_value
```

Description

Applies to the twoWidth table. It only returns the spacing value for a given width1, width2, and length.

Arguments

d_ConstraintID	The ID of the constraint.
n_index1	This is the first width or width1.
n_index2	This is the second width or width2.
n_index3	This should be the length.

Value Returned

n_value	The value of the given width1, width2, and length.

Example

```
twoWidthCst = cstCreateConstraint(foundry "minSpacing" list("metal1")
list(list(list("twoWidths" "length") 0.0)
list(list(0.0 -0.001) 0.15
list(0.0 0.0) 0.2
list(0.0 1.5) 0.5
list(0.0 3.0) 1.0
list(0.25 -0.001) 0.2
list(0.25 0.0) 0.25
list(0.25 1.5) 0.5
list(0.25 3.0) 1.0
list(1.5 -0.001) 0.5
list(1.5 0.0) 0.5
```

```
list(1.5 1.5) 0.6
list(1.5 3.0) 1.0
list(3.0 -0.001) 1.0
list(3.0 0.0) 1.0
list(3.0 1.5) 1.0
list(3.0 3.0) 1.2)) list(list('widthLengthTableType "twoWidthPRL")))
cstGetTwoWidthTableValue(twoWidthCst 0.25 0.25 0.1)
```

cstIsId

```
cstIsId(
     g_id
     )
     => t / nil
```

Description

Returns a Boolean that indicates whether the specified ID is a constraint ID.

Arguments

g_id	The ID of an object.

Value Returned

`t`	Returned if the given ID is a constraint ID.
`nil`	Returned if the given ID is not a constraint ID.

Example

```
cg1OnTopCV = cstCreateConstraintGroupIn(cv "cg1OnTopCV" "userDefined")
> cst:0x107ae616
cstIsId(nil)
> nil
cstIsId(cv)
> nil
cstIsId(cg1OnTopCV)
> t
```

Constraint Group Functions

The list of constraint group functions is given below:

- cstAddToConstraintGroup
- cstCreateConstraintGroupIn
- cstCreateConstraintGroupOn
- cstDeleteConstraintGroup
- cstFindConstraintGroupIn
- cstFindConstraintGroupOn
- cstFindFirstConstraint
- cstGetConstraintGroups
- cstGetDefaultConstraintGroupName
- cstSetDefaultConstraintGroupName

cstAddToConstraintGroup

```
cstAddToConstraintGroup(
     g_cstConstraintGroupID
     g_memberConstraintGroupID
     [ g_append ]
     )
     => t / nil
```

Description

Adds a constraint group to be a member constraint group of the containing constraint. You can position the constraint group at the beginning of the members or at the end of the members with `g_append`. The default value for `g_append` is `nil`, which means that add the constraint group to the beginning of the members. The member constraint group cannot be owned by other objects.

Note: A member constraint group cannot be added to itself.

Arguments

g_cstConstraintGroupID	The ID of the constraint group.
g_memberConstraintGroupID	The ID of the member constraint group.
g_append	A Boolean having a value as either nil or t. Having the value as nil puts this constraint to the first member in the constraint group. The present first constraint group member is moved to the second and subsequent members are also moved down in order. However, if the value is t, then the constraint is put to the end of the member in the constraint group. By default, the value is nil.

Value Returned

`t`	Puts the constraint to the end of the member in the constraint group.
`nil`	Puts the constraint at the beginning of the members in the constraint group.

Example

```
cstAddToConstraintGRoup (netcstGroup1 cvCstGroup1 nil)
```

Here, `nil` is for append argument.

```
cstAddToConstraintGroup(containingCG1 containedCG2 t)
```

cstCreateConstraintGroupIn

```
cstCreateConstraintGroupIn(
     { d_techID | d_cellViewID }
     t_constraintGroupName
     [ t_constraintGroupDefName ]
     [ g_operator ]
     )
     => g_cstConstraintGroupID / nil
```

Description

Creates a constraint group in the technology database (that is, `techDb`) or a cellview with arguments of the database ID, constraint group name, constraint group definition name and operator symbol.

Arguments

d_techID	The ID of the technology database.
d_cellViewID	The ID of the cellview.
t_constraintGroupName	The group name that needs to be specified by the user.
t_constraintGroupDefName	For technology database, the DFII constraint group definition name can be `"userDefined"`, `"foundry"`, `"implicit"`, `"default"`, and `"cutClass"`. However for cellview, the DFII constraint group definition name can be `"userDefined"`, `"implicit"`, and `"default"`. The default name is `"userDefined"`.
g_operator	The operator symbol can be `'precedence`, `'and` or `'or`. The default symbol is `'precedence`.

Value Returned

g_cstConstraintGroupID	Returns the ID of the constraint group, which is created in the technology database or a cellview.
`nil`	Returns nil if the constraint group is not created.

Example

Create a constraint group `techCstGroup1` in a techDB (techDB1).

```
techcstGroup1 = cstCreateConstraintGroupIn(techDB1 "techcstGroup1”).
techCG2 = cstCreateConstraintGroupIn(tech "spacingCG1" "userDefined" 'or)
```

Create a constraint group `cvcstGroup1` in a cellview (cv1).

```
cvcstGroup1 = cstCreateConstraintGroupIn(cv1 "cvcstGroup1”).
```

cstCreateConstraintGroupOn

```
cstCreateConstraintGroupOn(
     g_object
     t_constraintGroupDefName
     [ g_operator ]
     )
     => g_cstConstraintGroupID / nil
```

Description

Creates a constraint group on an object (for example, net, term, and so on) with arguments of the object ID, constraint group definition name, and operator symbol. The constraint group definition name is required while constraint group operators are optional for this SKILL function.

Arguments

g_object — An object in the design, such as net, route, and so on.

t_constraintGroupDefName — The DFII constraint group definition name can be as follows:

Constraint Group	Description	Objects
`implicit`	Specifies the semantics for the constraint group associated with individual objects.	all objects
`default`	Specifies the semantics for the constraint group associated with a container object. Constraints in these constraint groups apply to the contained objects but not to the container object.	Net, term, route, figGroup

`inputTaper`	Specifies the semantics for the input taper constraint group.	Net
`outputTaper`	Specifies the semantics for the output taper constraint group.	Net
`taper`	Specifies the semantics for the taper constraint group.	term, pin, instTerm
`shielding`	Specifies the semantics for the constraint group associated with a shielded bitNet. The shielding constraint group can also be associated with a group of nets that must be shielded.	Net
`transReflexive`	Specifies the semantics for constraint groups where the constraints apply between all objects within a container and all relevant objects outside of that container. The constraints do not apply between objects within the container.	group

	`reflexive`	Specifies the semantics for constraint groups where the constraints apply between the relevant objects within a container, but do not apply to objects outside of the container.	group
	`interChild`	Specifies the semantics for constraint groups where the constraints apply between objects within child containers but do not apply between objects within the parent container. InterChild constraint groups can be considered as applying to a subset of the relationships defined by the Reflexive semantics.	group
g_operator	Returns the ID of the constraint group that is created on an objects.		

Value Returned

g_cstConstraintGroupID	Returns the ID of the constraint group, which is created in the technology database or a cellview.
`nil`	Returns `nil` if the constraint group is not created on an object.

Example

```
netcstGroup1 = cstCreateConstraintGroupOn (net1 "inputTaper" "'precedence").
netCG2 = cstCreateConstraintGroupOn(net1 "outputTaper" 'or)
```

cstDeleteConstraintGroup

```
cstDeleteConstraintGroup(
     g_cstConstraintGroupID
     )
     => t / nil
```

Description

Deletes the constraint group (`g_cstConstraintGroupID`). The contained constraints in the constraint group are not automatically deleted when the constraintGroup is deleted.

Arguments

g_cstConstraintGroupID	The identification of the constraint group.

Value Returned

`t`	Returns `t` when the constraint group is successfully deleted.
`nil`	Returns `nil` when unable to find a constraint group that is to be deleted.

Example

```
cstDeleteConstraintGroup(cstGroupID)
```

cstFindConstraintGroupIn

```
cstFindConstraintGroupIn(
     { d_techID | d_cellViewID }
     t_constraintGroupName
     [ g_localOnly ]
     )
     => g_cstConstraintGroupID / nil
```

Description

Returns a constraint group ID (`g_cstConstraintGroupID`) from a database ID with constraint group name and a Boolean flag (`g_localOnly`).

Arguments

d_techID	The ID of the technology database.
d_cellViewID	The ID of the cellview.
t_constraintGroupName	The name of the constraint group.
g_localOnly	Looks for a constraint group in an ITDB graph if the database ID is a technology database.The default value is `nil`.

Value Returned

g_cstConstraintGroupID	Returns the ID of the constraint group that is found in the database.
`nil`	Returns `nil` if unable to find any constraint group in the database.

Example

```
foundryCGId = cstFindConstraintGroupIn(techDB1 "foundry" 't)
cv1CG1 = cstFindConstraintGroupIn(cellView1 "cvCG1")
```

cstFindConstraintGroupOn

```
cstFindConstraintGroupOn(
     g_object
     t_constraintGroupDefName
     )
     => g_cstConstraintGroupID / nil
```

Description

Returns a constraint group ID (`g_cstConstraintGroupID`) from an object (`g_object`) with constraint group definition name.

Arguments

g_object	An object in the design, such as net, route, and so on.
t_constraintGroupDefName	The DFII constraint group definition name. For information on this argument, refer to the cstCreateConstraintGroupOn function.

Value Returned

g_cstConstraintGroupID	Returns the ID of the constraint group that is found in an object.
`nil`	Returns `nil` if unable to find any constraint group in an object.

Example

```
netcstGroup1 = cstFindConstraintGroupOn (net1 "inputTaper").
```

cstFindFirstConstraint

```
cstFindFirstConstraint(
     d_ConstraintGroupID
     t_constraintDefName
     [ ( tx_layer [ tx_purpose ] )... ]
     [ l_params ]
     [ g_onlyHard ]
     )
     => d_ConstraintID / nil
```

Description

Searches a constraint group in the hierarchical order and returns the ID of the first constraint found with the given name, layers, purposes, and params. If the `onlyHard` argument is not specified, the first matching constraint is returned and it is possible that it is a hard constraint. If the `onlyHard` argument is specified, then the first matching hard constraint is returned.

Arguments

d_ConstraintGroupID	The ID of the constraintGroup in which the constraint is searched.
t_constraintDefName	Mapped DFII constraint name will be used to get a corresponding constraint definition name.
tx_layer...	List of layer names, layer numbers, or LPPs.

l_params	A SKILL list of constraint parameters. The default value is `nil`. The input `constraintParams` is in a format of:
	`list(list('paramKeyWord{int | float | string | list})...)`
	For example,
	`cst1~>params = list(list('ConnectivityType "sameNet") ('list('PGNet t)`
	`=>((connectivityType "sameNet") (PGNet t))`

g_onlyHard	If this argument is not specified, the first matching constraint is returned and it is possible that it is a hard constraint. If the argument is specified, then the first matching hard constraint is returned.

Value Returned

d_ConstraintID	Returns the ID of the constraint, if found.
`nil`	Returns `nil` if the matching constraint is not found.

Example

```
cstfound = cstFindFirstConstraint(foundry "minSpacing" list("Metal2"))
cstfound = cstFindFirstConstraint(foundry "minSpacing" list("Metal2") nil t)
cstfound = cstFindFirstConstraint(foundry "minSpacing" list(list("Metal1"
"drawing")) nil t)
```

cstGetConstraintGroups

```
cstGetConstraintGroups(
     g_cstConstraintID
     )
     => l_containingConstraintGroups / nil
```

Description

Returns all the containing constraint groups for the constraint (`g_cstConstraintID`).

Arguments

g_cstConstraintID	The identification of the constraint.

Value Returned

l_containingconstrainGroups	
	Returns all the constraint groups that contain the particular constraint.
`nil`	Returns `nil` when the particular constraint is not in any constraint groups.

Example

```
list_CGs = cstGetConstraintGroups(constraintId)
```

cstGetDefaultConstraintGroupName

```
cstGetDefaultConstraintGroupName(
     d_cellView
     t_type
     )
     => t_constraintGroupName / nil
```

Description

Gets the name of the constraint group that is used as the default constraint group on the objects of the cellview according to the specified application type.

Arguments

d_cellView	The cellview is allied with the specified constraint group and application type.
t_type	The four valid application types are "Setup", "Wire" "Via", and "Taper".

Value Returned

t_constrainGroupName	If the constraint group is in the cellview, this function returns `"dsn:`*`<constraintGroup name>`*". For example, `"dsn:default"` and `"dsn:cg1"`. If the cellview has no default constraint group, but its attached to the technology database that has a default constraint group, this function returns `"tech:default"`. If the constraint group is in the root technology database of the cellview, this function returns `"`*`<constraintGroup name>`*`"`. For example, `"foundry"`. If the constraint group is in the referenced technology database of the root technology database of the cellview, this function returns `"tech:`*`<constraintGroup name>`*`(`*`<reference techDB name>`*`"`. For example, `"tech:cg1(refTechDB2)"`.

`nil`	Returns `nil` when unable to find constraint group name.

Example

```
cstGetDefaultConstraintGroupName(cv1 "Wire")
=> "tech:default"
cstGetDefaultConstraintGroupName(cv2 "Wire")
=> "default"
cstGetDefaultConstraintGroupName(cv1 "Setup")
=> ""
cstGetDefaultConstraintGroupName(cv3 "Wire")
=> "tech:cg1(refTechDB2)"
```

cstGetFoundryCGName

```
cstGetFoundryCGName(
    )
    => t_constraintGroupName
```

Description

(ICADVM20.1 Only) Returns the name of the alternate foundry constraint group. If there is no alternate foundry constraint group specified, it returns the name of the foundry constraint group.

Arguments

None

Value Returned

t_constraintGroupName	The name of the alternate foundry constraint group (if specified) or the foundry constraint group.

Example

Returns the name of the alternate foundry constraint group specified:

```
cgName = cstGetFoundryCGName()
```

cstGetFoundryConstraintGroup

```
cstGetFoundryConstraintGroup(
     d_techID
     )
     => d_cgID / nil
```

Description

(ICADVM20.1 Only) Returns the database ID of the alternate foundry constraint group specified for a given technology file. If there is no alternate foundry constraint group specified, it returns the ID of the foundry constraint group.

Arguments

d_techID	Database ID of the technology file to be queried.

Value Returned

d_cgID	Database ID of the alternate foundry constraint group (if specified) or the foundry constraint group.
nil	Returns `nil` when the foundry constraint group cannot be returned due to the wrong argument or other error condition.

Example

Returns the alternate foundry constraint group specified for the technology file assigned to pointer `tfId`:

```
cgID = cstGetFoundryConstraintGroup(tfId)
```

cstSetDefaultConstraintGroupName

```
cstSetDefaultConstraintGroupName(
     d_cellview
     t_constraintGroupName
     t_type
     )
     => t / nil
```

Description

Sets the specified constraint group name to be used as the default constraint group on the objects of the cellview according to the specified application type.

Arguments

d_cellview	The cellview is allied with the specified constraint group and application type.
t_constraintGroupName	The name of the constraint group. If *t_constraintGroupName* has prefix `dsn:`, this function only finds the constraint group in the cellview with the string. If *t_constraintGroupName* has prefix `tech:`, this function only finds the constraint group from the technology database graph attached to the cellview with the string. If *t_constraintGroupName* has neither `dsn:` nor `tech:` as a prefix, this function finds the constraint group from the cellview first and then from the technology database graph attached to the cellview with the string
t_type	The four valid application types are "Setup", "Wire" "Via", and "Taper".

Value Returned

`t`	Returns `t` when the specified constraint group is used as the default constraint group.

`nil`	Returns `nil` when unable to use the specified constraint group as the default constraint group.

Example

```
cstSetDefaultConstraintGroupName(cv1 "default" "Wire")
cstSetDefaultConstraintGroupName(cv1 "dsn:default" "Wire")
cstSetDefaultConstraintGroupName(cv1 "tech:foundry" "Setup")
cstSetDefaultConstraintGroupName(cv1 "foundry" "Wire")
```

3

Design Intent Functions

This chapter covers details about the SKILL functions available to create and edit Design Intent. These functions are available only in the ICADVM20.1 release.

- ciDevGroupBoxIterator
- ciDiMinMaxVPropertyCallback
- ciDiPostTransferHighCurrent
- ciDiPostTransferMinMaxVoltage
- ciDiReplaceOrAddPropertyGroupDef
- ciDiReportGenReport
- ciSetDIPropertyGroupDefs
- ciTemplateChangeDIProfile
- ciTemplateCreateDI
- ciTemplateDIProfileName
- ciTemplateDIPropDef
- ciTemplateDIPropGroupDef
- ciTemplateDIPropValue
- ciTemplateIsKindOfDI
- ciTemplateListDIProps
- ciTemplateUpdateDIProps
- ciUpdateObjPropsFromDI

Related Topics

Virtuoso Design Intent User Guide

ciDevGroupBoxIterator

```
ciDevGroupBoxIterator(
     d_cellview
     t_finderExpr
     )
     => l_dbId / nil
```

Description

(ICADVM20.1 Only) Iterates all device groups within a cellview that are enclosed by a text box.

Arguments

d_cellview	The cellview containing the design instances to be iterated.
t_finderExpr	The matched expression string to be used in the iteration.

Value Returned

l_dbId	A list of lists, where each sublist contains the devices within each group.
nil	No text boxes were identified.

Example

```
ciDevGroupBoxIterator(geGetEditCellView() "ciIsDevice(inst \“fet\”) &&
!ciIgnoreDevice(inst)")~>name
(("M12" "M5" "M4")
    ("PM7")
    ("M2" "M3")
)
```

Three text boxes were identified, each containing a list of instances as shown.

ciDiMinMaxVPropertyCallback

```
ciDiMinMaxVPropertyCallback(
     g_field
     f_form
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Runs a callback to check that customized property definitions using minimum and maximum voltage property values are set correctly so that *Min Voltage* is less than or equal to *Max Voltage*. When these properties are set incorrectly in the Create Design Intent form or Edit Design Intent form, the properties are highlighted and the form is prevented from closing until the values are corrected.

Arguments

g_field	The design intent field associated with the minimum and maximum voltage properties. For example, *Max Voltage* and *Min Voltage* on the *Net Voltage* profile.
f_form	The Create Design Intent form or the Edit Design Intent form.

Value Returned

`t`	The minimum and maximum values are set correctly.
`nil`	The minimum and maximum values are incorrectly set. The properties are highlighted in the Create Design Intent form or Edit Design Intent form for the values to be adjusted.

Example

The callback checks whether the values are set correctly for the `Min Voltage` and `Max Voltage` properties and highlights any errors:

```
list(nil 'name "Net Voltage" 'category "Nets" 'toolTip "Net Min/Max Voltage"
    'properties list
```

```
            list(nil 'name "Min Voltage" 'type 'float 'defValue 0.0 'range '(0.0
                9999999.9) 'callback 'ciDiMinMaxVPropertyCallback)
            list(nil 'name "Max Voltage" 'type 'float 'defValue 0.0 'range '(0.0
            9999999.9) 'callback 'ciDiMinMaxVPropertyCallback)
        )
    )
```

Related topics

Creating a Design Intent

Defining Property Profiles

ciDiPostTransferHighCurrent

```
ciDiPostTransferHighCurrent(
     g_template
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Runs a callback to enslure that when a customized HighCurrent design intent is transferred from Schematics XL to Layout XL, the current is split evenly between the mfactored members in Layout XL. This function is specified using the `postTransferCallback` for the HighCurrent design intent properties.

Arguments

g_template	The design intent template.

Value Returned

`t`	The design intent properties were set using the specified design intent template.
`nil`	The design intent properties were not set.

Example

Runs `ciDiPostTransferHighCurrent` for the `I` property on the `Current` profile:

```
list(nil 'name "Current" 'category  "HighCurrent" 'toolTip "High Current Pins/
    Terminals"
        'properties list(list(nil 'name "I" 'type 'current) )
        'postTransferCallback 'ciDiPostTransferHighCurrent
    )
```

Related topics

Creating a HighCurrent Design Intent

Defining Property Profiles

ciDiPostTransferMinMaxVoltage

```
ciDiPostTransferMinMaxVoltage(
     g_template
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Runs a callback to propagate the design intent properties *Min Voltage*, *Max Voltage*, *Signal Type*, *Power Sensitivity*, and *Ground Sensitivity* on the associated design intent objects after transfer from schematic to layout.

For more details, see Defining Property Profiles.

Arguments

g_template	The transferred design intent template.

Value Returned

`t`	The design intent properties were set on the associated design intent objects.
`nil`	The design intent properties were not set.

Example

Propagates the design intent properties `Min Voltage` and `Max Voltage` on to the associated pins design intent after transfer from schematic to layout.

```
ciDiReplaceOrAddPropertyGroupDef(
    list(nil 'name "Pin Voltages" 'category "Pins" 'toolTip "Pin voltage rules"
        'properties list(
            list(nil 'name "Min Voltage" 'type 'float 'defValue 0.0 'range
                '(0.09999999.9) 'callback 'ciDiMinMaxVPropertyCallback)
            list(nil 'name "Max Voltage" 'type 'float 'defValue 0.0 'range
                '(0.09999999.9) 'callback 'ciDiMinMaxVPropertyCallback
            )
        'postTransferCallback 'ciDiPostTransferMinMaxVoltage
        'okApplyCallback 'ciUpdateObjPropsFromDI
    )
)
```

Related topics

Creating a MaxVoltageDrop Design Intent

Defining Property Profiles

ciDiReplaceOrAddPropertyGroupDef

```
ciDiReplaceOrAddPropertyGroupDef(
     l_propGroupDefs
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Adds or replaces customized design intent property group definitions.

For more details, see Defining Property Profiles.

Arguments

l_propGroupDefs	The design intent property group definition to be added or replaced.

Value Returned

`t`	The new or amended property group definition was successfully added or replaced.
`nil`	The property group definition was not added.

Example

Adds a new property group definition, `Pin Voltages`, to the design intent category, `Pins`:

```
ciDiReplaceOrAddPropertyGroupDef(
    list(nil 'name "Pin Voltages" 'category "Pins" 'toolTip "Pin voltage rules"
        'properties list(
            list(nil 'name "Min Voltage" 'type 'float 'defValue 0.0 'range '(0.0
            9999999.9) 'callback 'ciDiMinMaxVPropertyCallback)
            list(nil 'name "Max Voltage" 'type 'float 'defValue 0.0 'range '(0.0
            9999999.9) 'callback 'ciDiMinMaxVPropertyCallback)
        )
    )
)
```

ciDiReportGenReport

```
ciDiReportGenReport(
     [ ?cv d_cellviewID ]
     [ ?depth x_depth ]
     [ ?title t_reportName ]
     [ ?path t_path ]
     [ ?launch { t | nil } ]
     [ ?date t_reportDate ]
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Generates a report summarizing the design intent in the current design based on the specified criteria and optionally opens the report in the browser.

Arguments

`?cv` *d_cellviewID*	Database ID of the cellview for which the report is to be generated. The default is the cellview currently being edited.
`?depth` *x_depth*	Number of hierarchy levels to be included in the report. The default is `-1`, which means that the report is generated for the full design hierarchy. Specify `0` to generate a report only for the specified cellview.
`?title` *t_reportName*	Title of the report. If you do not specify a name, the system generates one automatically.
`?path` *t_path*	Path to the directory in which the report is saved. If you do not specify a path, the report is saved in the current working directory.
`?launch { t | nil }`	Automatically opens the report in the browser after it is generated.
`?date` *t_reportDate*	Date to be appended to the report. If you do not specify a date, the date and time the report was generated are used.

Value Returned

`t`	The report was generated and saved to disk.
`nil`	The report was not generated.

Example

Generates a report for the top-level cellview currently being edited. The title of the report is Design Intent Report, it is to be saved to a `reports` directory inside the current working directory, and it will be opened automatically in the browser after it is generated.

```
(ciDiReportGenReport ?cv (geGetEditCellView) ?depth 0 ?title "Design Intent Report"
?path "./reports" ?launch t)
```

ciSetDIPropertyGroupDefs

```
ciSetDIPropertyGroupDefs(
     l_propGroupDefs
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Sets the design intent property group definitions to the specified definitions. All existing property group definitions are deleted.

For more details, see Defining Property Profiles.

Arguments

l_propGroupDefs	A list of design intent property group definitions.

Value Returned

`t`	The property group definitions were replaced.
`nil`	The property group definitions were not set.

Example

All existing property group definitions are replaced with the design intent property group definitions `Net Voltages` and `Pin Voltages`. The callback `ciDiMinMaxVPropertyCallback` is run to check the values for the minimum and maximum voltage properties for `minVProp` and `maxVProp`:

```
let(((minVProp list(nil 'name "Min Voltage" 'type 'float 'defValue 0.0 'range '
        (0.0 9999999.9) 'callback 'ciDiMinMaxVPropertyCallback))
     (maxVProp list(nil 'name "Max Voltage" 'type 'float 'defValue 0.0 'range '
        (0.0 9999999.9) 'callback 'ciDiMinMaxVPropertyCallback)))

ciSetDIPropertyGroupDefs( list(
    list(nil 'name "Net Voltages" 'category "Nets" 'toolTip "Net voltage rules"
        'properties list(
            minVProp
            maxVProp
            )
        )
    )
```

```
    list(nil 'name "Pin Voltages" 'category "Pins" 'toolTip "Pin voltage rules"
        'properties list(
            minVProp
            maxVProp
            )
        )
    )
)
```

ciTemplateChangeDIProfile

```
ciTemplateChangeDIProfile(
     u_templateId
     t_profileName
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Replaces the property profile currently selected for a design intent template.

Arguments

u_templateId	The design intent template ID.
t_profileName	The new profile name.

Value Returned

t	The property profile was replaced successfully.
nil	The property profile was not replaced.

Example

```
firstDeviceDI = car(setof(plate ciGetCellView()->templates plate->type ==
"diDevices"))

ciTemplateDIProfileName(firstDeviceDI)
"Match - default"

ciTemplateListDIProps(firstDeviceDI)
(("FoldDevice" string "Even")
    ("ShareDiffusion" boolean nil)
    ("Surround" string "Dummy")
    ("PlacementStyle" string "Symmetry")
)

ciTemplateChangeDIProfile(
    firstDeviceDI "Match - DiffPair"
)

ciTemplateDIProfileName(firstDeviceDI)
"Match - DiffPair"
```

```
ciTemplateListDIProps(firstDeviceDI)
(("MatchNetPair" string "")
    ("WPERange" float 0.1)
    ("FoldDevice" string "Even")
    ("Surround" string "Dummy")
    ("ShareDiffusion" boolean nil)
    ("PlacementStyle" string "Symmetry")
)
```

For a design intent template of type `diDevices`, this example uses the function `ciTemplateDIProfileName` to return the profile currently selected for the design intent template, `Match - default`. The function `ciTemplateListDIProps` is then used to return the property names, types, and values associated with it.

Using the `ciTemplateChangeDIProfile` function, the profile currently selected for the template is then changed to `Match - DiffPair`. The `ciTemplateListDIProps` function is used again to return the property names, types, and values associated with the new profile.

ciTemplateCreateDI

```
ciTemplateCreateDI(
     g_cache
     t_designIntentTemplateType { diDevices | diNets | diPins | diMaxVoltageDrop
     | diHighCurrent | diCell }
     t_name
     l_memberNamesTypes
     [ ?profile t_profileName ]
     [ ?params l_paramNameAndValue ]
     [ ?propNameValue l_propNameValue ]
     )
     => u_templateId / nil
```

Description

(ICADVM20.1 Only) Creates a design intent template for the specified template type.

Arguments

g_cache	The constraints cache.
t_designIntentTemplateType	The design intent template type enclosed in straight quotes. The types available are `diDevices`, `diNets`, `diPins`, `diMaxVoltageDrop`, `diHighCurrent`, and `diCell`.
t_name	The name of the created template.
l_memberNamesTypes	The member names and types available to the template created.
`?profile` *t_profileName*	
	The profile name to be associated with the new template. Each profile must be enclosed in quotation marks.
`?params` *l_paramNameAndValue*	
	A list of the parameter names and values. Valid parameter names are `Design Notes`, `Color`, `AnnotationStyle`, `Font Height`, `Font Style` and `Signed Off`.

`?propNameValue` *`l_propNameValue`*

The individual property types to be included in the new template. Each property type must be enclosed in quotation marks.

Value Returned

`u_templateId`	The ID of the template created.
`nil`	A template was not created.

Example

```
cache = ciGetCellView()
diDevices = ciTemplateCreateDI(cache "diDevices" "DiffPair" '(("M2" inst)("M3"
inst)) ?profile "Match - DiffPair" ?params '(("Design Notes" "AABB/BBAA")))

ciTemplateUpdateParams(
    diDevices '(("Design Notes"
    "ABBA/BAAB")
    ("AnnotationStyle" "BoundingBoxes")
    ("Color" "halo4"))) ;; halo packets used for color

diNets = ciTemplateCreateDI(cache "diNets" "Match Nets" '(("INN" net)("INP" net))
?profile "Match Net")

diPins = ciTemplateCreateDI(cache "diPins" "Diff Pair Pins" '(("INN:3" pin)("INP:4"
pin)) ?profile "Placement")

diMVD = ciTemplateCreateDI(
    cache
    "diMaxVoltageDrop"
     "Max Voltage Drop (Power)"
    '(
    ("VDD:2" pin      (("ref" 1)("enableV" 1)))
    ("PM0:S" instTerm (("ref" 0)("enableV" 1)("Voltage" 0.11)))
    ("PM7:S" instTerm (("ref" 0)("enableV" 1)("Voltage" 0.22)))
    )
    ?profile "Voltage") ;;; Must be this profile name

diHighC = ciTemplateCreateDI(
```

```
    cache
    "diHighCurrent"
    "High Current Paths"
    '(
    ("VSS:1" pin      (("enableI" 1)("Current" 0.75)))
    ("NM7:S" instTerm (("enableI" 1)("Current" 0.11)))
    ("NM8:S" instTerm (("enableI" 1)("Current" 0.22)))
    ("NM9:S" instTerm (("enableI" 1)("Current" 0.33)))
    )
    ?profile "Current") ;;; Must be this profile name

diCell = ciTemplateCreateDI(cache "diCell" "Cell Level DI" '(("ether_adc45n/
adc_sample_hold/schematic" " 'master)))
```

This example creates the following design intents for a cellview:

1. `DiffPair` based on the template type `diDevices` is created with the property profile `Match - DiffPair` for the instances `M2` and `M3`. The annotation style and notes parameters are also specified for the design intent.
2. `Match Nets` is created based on the template type `diNets` with the property profile `Match Net` for the nets `INN` and `INP`.
3. `Diff Pair Pins` based on the template type `diPins` is created with the property profile `Placement` for the pins `INN:3` and `INP:4`.
4. `Max Voltage Drop (Power)`, based on the template type `diMaxVoltageDrop`, it is created with the property profile `Voltage` for the pin `VDD:2` and instance terminals `PMO:S` and `PM7:S`. The profile property parameters are specified for this design intent.
5. `High Current Paths`, based on the template type `diHighCurrent`, it is created with the property profile `Current` for the pin `VDD:1` and instance terminals `NM7:S`, `NM8:S`, and `NM9:S`. The profile property parameters are specified for this design intent.
6. `Cell Level DI`, based on the template type `diCell` which is a design intent that applies to the entire cellview.

ciTemplateDIProfileName

```
ciTemplateDIProfileName(
     u_templateId
     )
     => t_profileName / nil
```

Description

(ICADVM20.1 Only) Returns the name of the profile associated with the specified design intent template.

Arguments

u_templateId	The template ID where the property profiles are located.

Value Returned

t_profileName	The profile name associated with the specified template.
nil	No property profiles were located.

Example

```
foreach(mapcar plate ciGetCellView()->templates list(plate->type
ciTemplateDIProfileName(plate)))
   (("diDevices" "Match - CurrentMirror")
    ("match" nil)
    ("diNets" "Match Net")
    ("commonCentroid" nil)
)
```

Lists all the templates in the cellview and for those that are design intent templates, returns the profile name.

ciTemplateDIPropDef

```
ciTemplateDIPropDef(
     u_templateId
     t_propertyName
     )
     => l_propNameTypeDefVal / nil
```

Description

(ICADVM20.1 Only) For the specified design intent template, returns the definition of the named property in the form of a DPL.

Arguments

u_templateId	The template ID where the named property is located.
t_propertyName	Specifies the name of the property for which the definition is to be returned.

Value Returned

l_propNameTypeDefVal	The property definition in DPL form.
nil	No matching property definition was found.

Example

```
foreach(mapcar plate ciGetCellView()->templates list(plate->type
ciTemplateDIPropDef(plate "PlacementStyle")))
(("diDevices"
    (nil name "PlacementStyle" type enum
    defValue "Symmetry" items
    ("Symmetry" "CommonCentroid" "Interdigitate")
    )
    )
("match" nil)
("diNets" nil)
("commonCentroid" nil)
)
```

Lists all the templates in the cellview and for those that are design intent templates, returns the definition of the property `PlacementStyle`.

ciTemplateDIPropGroupDef

```
ciTemplateDIPropGroupDef(
     u_templateId
     )
     => l_propNameTypeDefVal / nil
```

Description

(ICADVM20.1 Only) For the specified design intent template, returns the definition of all the properties in the form of a DPL.

Arguments

u_templateId	The ID of the template where the properties are located.

Value Returned:

l_propNameTypeDefVal	The property definition in DPL form.
nil	No properties were located.

Example

```
foreach(mapcar plate ciGetCellView()->templates list(plate->type
ciTemplateDIPropGroupDef(plate)))
(("diDevices"
    (nil name "Match - CurrentMirror" category "Devices"
    toolTip "How CurrentMirror devices are to match" properties
    ((nil name "PlacementStyle" type enum
        defValue "Symmetry" items
        ("Symmetry" "CommonCentroid" "Interdigitate")
        )
        (nil name "MatchProperty" type enum
        defValue "Parameter" items
        ("Parameter" "Orientation")
        )
        (nil name "Surround" type enum
        defValue "Dummy" items
        ("Dummy" "FGR" "Both" "asNeeded")
```

```
        )
        (nil name "ShareDiffusion" type bool
        defValue nil
        )
    )
    )
    ("match" nil)
    ("diNets" nil)
    ("commonCentroid" nil)
)
```

Lists all the templates in the cellview and for those that are design intent templates, returns the definition of each property.

ciTemplateDIPropValue

```
ciTemplateDIPropValue(
     u_templateId
     t_propertyName
     )
     => g_propValue / nil
```

Description

(ICADVM20.1 Only) For the specified design intent template, returns the value of a named property.

Arguments

u_templateId	The template ID where the named property is lcoated.
t_propertyName	The name of the property from which the value is to be returned.

Value Returned

g_propValue	The property value.
nil	No matching property was located.

Example

```
firstDeviceDI = car(setof(plate ciGetCellView()->templates plate->type ==
"diDevices"))
ciTemplateDIPropValue(firstDeviceDI "PlacementStyle") "CommonCentroid"
```

Finds the first design intent template on the cellview and returns the value for the named property `PlacementStyle`.

ciTemplateIsKindOfDI

```
ciTemplateIsKindOfDI(
     d_templateId
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Confirms if the specified constraint template is a design intent template. The available design intent templates are: *diDevices*, *diNets*, *diPins*, *diMaxVoltageDrop*, *diHighCurrent*, and *diCell*.

Arguments

d_templateId	The template ID to be checked.

Value Returned

t	The template ID is a design intent template.
nil	The template ID is not a design intent template

Example

ciTemplateIsKindOfDI

```
cache = ciGetCellView()
foreach(mapcar plate cache->templates list(plate->type
ciTemplateIsKindOfDI(plate)))
(("diDevices" t)
     ("match" nil)
     ("diNets" t)
     ("commonCentroid" nil)
)
```

Lists all the template IDs in the cellview and for the templates that are design intent templates, returns `t`.

ciTemplateListDIProps

```
ciTemplateListDIProps(
     u_templateId
     )
     => l_propNameTypeValueList / nil
```

Description

(ICADVM20.1 Only) For the specified design intent template, returns a list of the property names, types, and values.

Note: If a scope is defined for the design intent, this function automatically includes within the list returned, an additional syntax string `'name "DI_CURRENT_SCOPE"`. This lists the currently selected scope for the design intent as the `'defValue`. For more details, see Defining Scopes for Profile Properties.

Arguments

u_templateId	The template ID where the property names, types, and values are located.

Value Returned

l_propNameTypeValueList	Lists the property names, types, and values.
`nil`	No properties were located.

Example

```
foreach(mapcar plate ciGetCellView()->templates list(plate->type
ciTemplateListDIProps(plate)))
(("diDevices"
    (("ShareDiffusion" boolean nil)
        ("Surround" string "FGR")
        ("MatchProperty" string "Parameter")
        ("PlacementStyle" string "CommonCentroid")
    )
    )
    ("match" nil)
    ("diNets"
    (("Keep Away" string "")
```

```
        ("Shield" string "Parallel")
        ("Signal Type" string "Symmetry")
        ("Match Pair" string "")
    )
    )
    ("commonCentroid" nil)
)
```

Lists all the templates in the cellview and for those that are design intent templates, returns the name, type, and value of each property.

ciTemplateUpdateDIProps

```
ciTemplateUpdateDIProps(
     u_templateId
     l_propNameValue
     )
     => t / nil
```

Description

(ICADVM20.1 Only) For the specified design intent template, updates the value of the named property.

Arguments

u_templateId	The template ID where the named property is located.
l_propNameValue	The property name value to be updated.

Value Returned

`t`	The specified property value was updated.
`nil`	The specified property name was not found.

Example

```
firstDeviceDI = car(setof(plate ciGetCellView()->templates plate->type ==
"diDevices"))
ciTemplateDIPropValue(firstDeviceDI "PlacementStyle")
"CommonCentroid"
ciTemplateDIPropValue(firstDeviceDI "Surround")
"FGR"
ciTemplateUpdateDIProps(firstDeviceDI '(("PlacementStyle"
"Interdigitate")("Surround" "Dummy")))
ciTemplateDIPropValue(firstDeviceDI "PlacementStyle")
"Interdigitate"
ciTemplateDIPropValue(firstDeviceDI "Surround")
"Dummy"
```

Finds the first design intent template on the cellview and returns the values for the properties `PlacementStyle` and `Surround`. For `PlacementStyle`, the value `CommonCentroid` is replaced with `Interdigitate`, and for `Surround`, `FGR` is replaced with `Dummy`.

ciUpdateObjPropsFromDI

```
ciUpdateObjPropsFromDI(
     g_form
     g_template
     )
     => t / nil
```

Description

(ICADVM20.1 Only) Runs a callback to propagate design intent properties to the associated design intent objects. The function can be called when the Create Design Intent form or Edit Design Intent form are submitted.

For more details, see Defining Property Profiles.

Arguments

g_form	The Create Design Intent form or Edit Design Intent form.
g_template	The design intent constraint template.

Value Returned

`t`	The design intent properties were updated for all objects associated with the design intent.
`nil`	The design intent properties were not updated.

Example

Propagates the design intent properties `Min Voltage` and `Max Voltage` to the associated design net and `Signal Type` to pins.

```
list(nil 'name "Net Voltage" 'category "Nets" 'toolTip "Net Min/Max Voltage"
         'properties list(
            list(nil 'name "Min Voltage" 'type 'float 'defValue 0.0 'range
                '(0.0 9999999.9) 'callback 'ciDiMinMaxVPropertyCallback)
                list(nil 'name "Max Voltage" 'type 'float 'defValue 0.0 'range
                '(0.0 9999999.9) 'callback 'ciDiMinMaxVPropertyCallback)
             )
            'okApplyCallback      'ciUpdateObjPropsFromDI
    )

list(nil 'name "Signal Type" 'category "Pins" 'toolTip "Signal Type"
         'properties list(
```

```
        list(nil 'name "Signal Type" 'type 'enum  'defValue "signal" 'items
        '("signal" "ground" "power" "clock" "analog" "tieOff" "tieHi" "tieL"
          "scan" "reset"))
         )
        'okApplyCallback      'ciUpdateObjPropsFromDI
)
```