Cadence Application Infrastructure SKILL Reference

Product Version ICADVM20.1
October 2020

Contents

Preface

This document provides information about the Cadence® SKILL APIs to support consistent operations between applications. It also provides information on customize the Library Manager using the Cadence SKILL language functions.

In addition this document provides syntax, descriptions, and examples for the Cadence SKILL functions associated with the configuration of plotters.

Important

Only the functions and arguments described in this manual are available for public use. Any undocumented functions or arguments are likely to be private and could be subject to change without notice. It is recommended that you check with your Cadence representative before using them.

This user 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.

This preface contains the following topics:

- Scope
- Licensing Requirements
- Related Documentation
- Additional Learning Resources
- Customer Support
- Feedback about Documentation
- Understanding Cadence SKILL

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

Scope

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

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

Licensing Requirements

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

Related Documentation

What's New and KPNS

- *Cadence Application Infrastructure What's New*
- *Cadence Application Infrastructure Known Problems and Solutions*

Installation, Environment, and Infrastructure

- *Cadence Application Infrastructure User Guide*
- *Cadence Library Manager User Guide*
- *Plotter Configuration User Guide*

Virtuoso Tools

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

Additional Learning Resources

Video Library

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

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

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

Virtuoso Videos Book

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

Rapid Adoption Kits

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

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

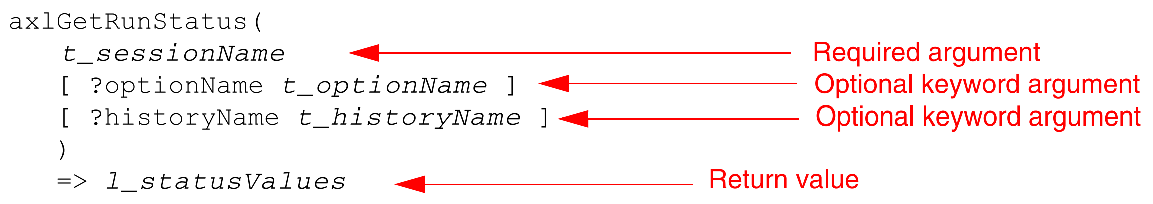
The SKILL APIs in this user manual are explained with illustrative code examples.

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

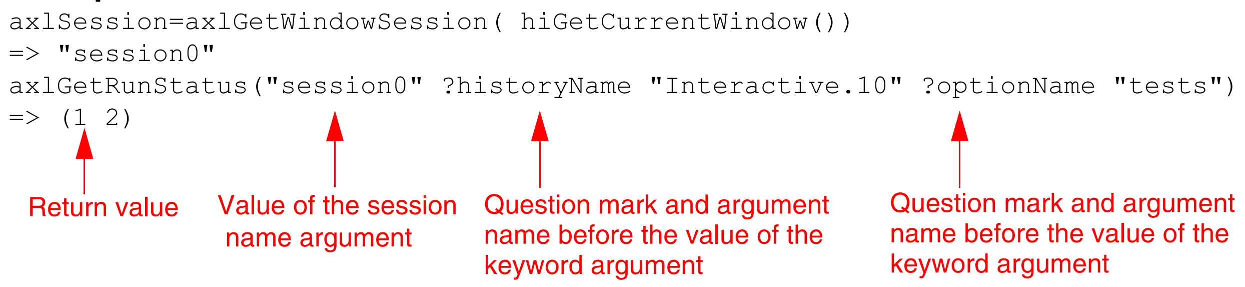


```
axlGetRunStatus(
    t_sessionName
    [ ?optionName t_optionName ]
    [ ?historyName t_historyName ]
    )
    => l_statusValues
```

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

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

Example



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

Return value
Value of the session name argument
Question mark and argument name before the value of the keyword argument
Question mark and argument name before the value of the keyword argument

Accessing API Help

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

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

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

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

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

Typographic and Syntax Conventions

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

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

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

Identifiers Used to Denote Data Types

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

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

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

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

1 Generic Design Management (GDM) Functions

This chapter provides information about Generic Design Management SKILL APIs that has a Cadence SKILL language interface. GDM SKILL functions have the prefix `gdm`.

You can use GDM functions to

- Perform the same operations as GDM shell commands, such as `gdmco` or `gdmci`
- Get gdmSpec objects of design data

  A gdmSpec object is a user-defined type of SKILL object. These objects can be used with any SKILL functions that need gdmSpec objects. For example, the cdsCopy SKILL functions described below, need library objects in the form of gdmSpec objects.

GDM has the following SKILL functions:

- gdmCreateSpec
- gdmCreateSpecFromDDID
- gdmInspectSpec
- gdmCreateSpecList
- gdmAddSpecToSpecList
- gdmResetSpecList
- gdmNextFromSpecList
- gdmIsSpecId
- gdmSpecType
- gdmSpecListp
- gdmSpecp
- gdmcancel

- gdmci
- gdmco
- gdmdelete
- gdmexport
- gdmhistory
- gdmsetdefver
- gdmsetname
- gdmstatus
- gdmsubmit
- gdmupdate
- gdmExecute
- gdmRemovename
- gdmObjIsCreated

Prefixes Used in Argument Names

Argument names in the SKILL functions described in this section have a one-character prefix that denotes the data type of the argument. The prefix is followed by an underscore character (_). For example, `t_libName` indicates the argument is a string, while `l_comps` indicates the argument is a list. The prefix and the underscore are just naming conventions; you do not need to type them when you specify the value of the argument.

The following data type prefixes are used in this section:

`t`	string
`x`	integer
`l`	list
`g`	general
`b`	ddId (Identifier of an object in the Cadence design library structure)
`G`	gdmSpec object
`q`	gdmSpecList object

Additional Information

For information about the SKILL language, refer to the following manuals:

- *Cadence SKILL Language User Guide*
- *Cadence SKILL Language Reference*
- *Cadence SKILL Development Help*
- *Cadence SKILL Development Reference*
- *Cadence SKILL++ Object System Reference*
- *Cadence Interprocess Communications SKILL Reference*
- *Cadence User Interface SKILL Reference*

gdmCreateSpec

```
gdmCreateSpec(
     t_libName | emptyString | nil
     t_cellName | emptyString | nil
     t_viewName | emptyString | nil
     t_fileName | emptyString | nil
     t_namespace
     [ x_gdmOptions ]
     )
     => G_gdmSpecId / nil
```

Description

Creates a gdmSpec object, a user-defined type of SKILL object, according to the options you specify. You must specify at least one of the first four arguments—a library name, cell name, view name, or file name. If you specify a cell name, you must also specify a library name. If you specify a view name, you must also specify a cell name and a library name.

Arguments

t_libName	Library name (string), an empty string (`""`), or `nil`.
t_cellName	Cell name (string), an empty string (`""`), or `nil`. If you specify a cell name, you must also specify a library name.
t_viewName	View name (string), an empty string (`""`), or `nil`. If you specify a view name, you must also specify a cell name and a library name.
t_fileName	File name (string), an empty string (`""`), or `nil`. **Note:** All the above four arguments cannot be empty strings; you must provide a name for at least one of them.
t_namespace	Name space in which the gdmSpec is to be created. Can be one of the following strings: `VHDL`, `VHDLAMS`, `Verilog`, `VerilogA`, `VerilogAMS`, `CDBA`, `Concept`, `Library`, `LibraryUnix`, `LibraryNT`, `Spectre`, `SpectreHDL`, or `Spice`. For more information about Cadence name spaces, see Name Spaces for Different Data Types.
x_gdmOptions	The following integer: `1`: Specifies that commands such as check out or check in work recursively on the directory. Applies only to gdmSpec objects that consist of directories.

Value Returned

G_gdmSpecId	The ID of the gdmSpec object created.
`nil`	The gdmSpec object could not be created.

Example

```
newSpec = gdmCreateSpec( "myLib" "" "" "" "CDBA" )
```

gdmCreateSpecFromDDID

```
gdmCreateSpecFromDDID(
     b_ddId
     )
     => G_gdmSpecId / nil
```

Description

Creates a gdmSpec, a user-defined type of SKILL object, from a ddId object.

Arguments

b_ddId	The ddId of the object for which you want to create a gdmSpec object.

Value Returned

G_gdmSpecId	The ID of the gdmSpec object created.
`nil`	The gdmSpec object could not be created.

gdmInspectSpec

```
gdmInspectSpec(
     G_gdmSpecId
     [ t_namespace ]
     )
     => l_comps
```

Description

From a gdmSpec object, extracts and returns the library name, cell name, view name, and file name, if they exist in the gdmSpec object. The information returned will be in the name space you specify in `t_namespace`.

Arguments

G_gdmSpecId	The gdmSpec of the object whose library name, cell name, view name, and file name you want to get.
t_namespace	Name space in which the return information is provided. Can be one of the following strings: `VHDL`, `VHDLAMS`, `Verilog`, `VerilogA`, `VerilogAMS`, `CDBA`, `Concept`, `Library`, `LibraryUnix`, `LibraryNT`, `Spectre, SpectreHDL`, or `Spice`. This argument is optional; if you do not provide a value, `CDBA` is used as the default name space. For more information about Cadence name spaces, see Name Spaces for Different Data Types.

Value Returned

l_comps	A list that contains the library name, cell name, view name, and file name, in that order. If any of these elements did not exist in the gdmSpec object, it is represented by `nil` in the return list.

gdmCreateSpecList

```
gdmCreateSpecList(
     )
     => q_gdmSpecList / nil
```

Description

Creates a gdmSpecList object, to which you can add gdmSpec objects later with the `gdmAddSpecToSpecList` function.

To traverse the gdmSpecList object,

1. Reset the list by using the `gdmResetSpecList` function.
2. Get gdmSpec objects by using the `gdmNextFromSpecList` function. The first time you call this function, it returns the first gdmSpec object from the gdmSpecList. Each successive call returns the next gdmSpec object from the gdmSpecList.

Arguments

This function does not take any arguments.

Value Returned

q_gdmSpecList	The gdmSpecList object created.
`nil`	The gdmSpecList object could not be created.

Example

You can traverse a gdmSpecList object in the following way:

```
specList = gdmCreateSpecList()
spec = gdmCreateSpec( "mylib" "topcell" "schematic" nil "CDBA" )
gdmAddSpecToSpecList( spec specList )
gdmResetSpecList( specList )
while( nextSpec = gdmNextFromSpecList( specList )
       println( gdmInspectSpec( nextSpec "CDBA" ))
     )
```

gdmAddSpecToSpecList

```
gdmAddSpecToSpecList(
     G_gdmSpec
     q_gdmSpecList
     )
     => t / nil
```

Description

Adds a gdmSpec object to a gdmSpecList object. This function automatically increases the size of the gdmSpecList object so that more gdmSpec objects can be added, if required.

Arguments

G_gdmSpec	The gdmSpec object you want to add to the gdmSpecList object.
q_gdmSpecList	The gdmSpecList to which you want to add a gdmSpec object.

Value Returned

`t`	The gdmSpec object was added to the gdmSpecList.
`nil`	The gdmSpec object could not be added to the gdmSpecList.

gdmResetSpecList

```
gdmResetSpecList(
     q_gdmSpecList
     )
     => t / nil
```

Description

Resets the gdmSpecList so that you can obtain gdmSpec objects from it with successive calls of the `gdmNextFromSpecList` function.

Use this function before you use the `gdmNextFromSpecList` function. If you reset the gdmSpecList and then call `gdmNextFromSpecList`, the first gdmSpec object in the gdmSpecList is returned.

For an example of how to use these functions, see "Example" on page 24.

Arguments

`q_gdmSpecList`	The gdmSpecList object you want to reset.

Value Returned

`t`	The gdmSpecList was reset.
`nil`	The gdmSpecList could not be reset.

gdmNextFromSpecList

```
gdmNextFromSpecList(
     q_gdmSpecList
     )
     => G_gdmSpecId / nil
```

Description

Takes a gdmSpecList object and returns a gdmSpec object from it. The first time you call this function, it returns the first gdmSpec object in the gdmSpecList. Each successive call gets the next gdmSpec object.

Note: Use this function with the gdmResetSpecList function. You *must* call `gdmResetSpecList` before the first call to `gdmNextFromSpecList`, otherwise the results may not be accurate.

For an example of how to use these functions, see "Example" on page 24.

Arguments

q_gdmSpecList	The `gdmSpecList` object from which the function returns gdmSpec objects. This object is the return value of the `gdmCreateSpecList` function.

Value Returned

G_gdmSpecId	The next gdmSpec object from the gdmSpecList object.
`nil`	The function failed.

gdmIsSpecId

```
gdmIsSpecId(
     g_object
     )
     => t / nil
```

Description

Checks whether an object is a valid gdmSpec object.

Arguments

g_object	The object you want to check.

Value Returned

`t`	*g_object* is a valid gdmSpec object.
`nil`	*g_object* is not a valid gdmSpec object.

gdmSpecType

```
gdmSpecType(
     G_gdmSpecId
     )
     => t_specType / nil
```

Description

Returns the type of a gdmSpec object. A gdmSpec object can be one of the following types: `lib`, `libCell`, `libCellView`, `libFile`, `libCellFile`, `libCellViewFile`, `directory`, or `file`.

Arguments

`G_gdmSpecId`	The gdmSpec object whose type you want to check.

Value Returned

`t_specType`	One of the following strings:
	`"lib"`
	`"libCell"`
	`"libCellView"`
	`"libFile"`
	`"libCellFile"`
	`"libCellViewFile"`
	`"directory"`
	`"file"`
`nil`	*`G_gdmSpecId`* is not a valid gdmSpec object.

gdmSpecListp

```
gdmSpecListp(
     g_object
     )
     => t / nil
```

Description

Checks whether an object is a `gdmSpecList`.

Arguments

`g_object`	The object you want to check.

Value Returned

`t`	*`g_object`* is a gdmSpecList.
`nil`	*`g_object`* is not a gdmSpecList.

gdmSpecp

```
gdmSpecp(
     g_object
     )
     => t / nil
```

Description

Checks whether an object is of type `gdmSpec`, a user-defined type of SKILL object.

Arguments

`g_object`	The object you want to check.

Value Returned

`t`	*`g_object`* is a gdmSpec object.
`nil`	*`g_object`* is not a gdmSpec object.

gdmcancel

```
gdmcancel(
     { G_gdmSpec | q_gdmSpecList }
     [ x_gdmOptions ]
     [ g_xtra ]
     )
     => t / nil
```

Description

Cancels the checked-out status of the library, cell, view, directory, or file that *`G_gdmSpec`* represents. Co-managed files in a view are always canceled as a group; co-managed set behavior applies only if *`G_gdmSpec`* consists of library elements.

If *`G_gdmSpec`* represents a directory that is not a library element, this function works recursively on the directory only if the `GDM_RECURSE` option was set when *`G_gdmSpec`* was created with gdmCreateSpec.

This function is the SKILL equivalent of the `gdmcancel` command. See `gdmcancel` for more information.

Arguments

`G_gdmSpec`	The gdmSpec object of the library, cell, view, directory, or file whose check-out you want to cancel.
`q_gdmSpecList`	A gdmSpecList object containing gdmSpec objects.
`x_gdmOptions`	This argument is currently ignored. The default is the integer `0`.
`g_xtra`	String containing additional arguments to be passed to the underlying design management system's cancel command.

Value Returned

`t`	The check-out status of the files was canceled.
`nil`	An error occurred and the check-out status of the files could not be canceled.

gdmci

```
gdmci(
     { G_gdmSpec | q_gdmSpecList }
     [ g_description ]
     [ x_gdmOptions ]
     [ g_xtra ]
     )
     => t / nil
```

Description

Checks in the library, cell, view, directory, or file that *G_gdmSpec* represents. Co-managed files in a view are always checked in as a group; co-managed set behavior applies only if *G_gdmSpec* consists of library elements.

If *G_gdmSpec* represents a directory that is not a library element, this function works recursively on the directory only if the `GDM_RECURSE` option was set when *G_gdmSpec* was created with gdmCreateSpec.

This function is the SKILL equivalent of the `gdmci` command. See `gdmci` for more information.

Arguments

G_gdmSpec	The gdmSpec object of the library, cell, view, directory, or file to check in.
q_gdmSpecList	A gdmSpecList object containing gdmSpec objects.
g_description	A description of the files that are checked in; specified as a string.
x_gdmOptions	One of the following integers: `5`: Optimized checks in previously unmanaged files for the first time. `4`: Optimized checks in already managed files. `1`: Checks in previously unmanaged files for the first time. `0`: Indicates no options are set. Default.
g_xtra	String containing additional arguments to be passed to the underlying design management system's check in command.

Value Returned

`t`	The files were checked in.
`nil`	An error occurred and the files were not checked in.

gdmco

```
gdmco(
     { G_gdmSpec | q_gdmSpecList }
     [ g_version ]
     [ x_gdmOptions ]
     [ g_xtra ]
     )
     => t / nil
```

Description

Checks out the library, cell, view, directory, or file that *G_gdmSpec* represents. Co-managed files in a view are always checked out as a group; co-managed set behavior applies only if *G_gdmSpec* consists of library elements.

If *G_gdmSpec* represents a directory that is not a library element, this function works recursively on the directory only if the `GDM_RECURSE` option was set when *G_gdmSpec* was created with gdmCreateSpec.

This function is the SKILL equivalent of the `gdmco` command. See `gdmcocom` for more information.

Arguments

G_gdmSpec	The gdmSpec object of the library, cell, view, directory, or file to check out.
q_gdmSpecList	A gdmSpecList object containing gdmSpec objects.
g_version	The version of the files to check out; specified as a string. You can specify this argument only if *G_gdmSpec* represents a file or a view.
x_gdmOptions	One of the following integers: `1`: Checks out all managed view files instead of only co-managed view files. `0`: Indicates no options are set. Default.
g_xtra	String containing additional arguments to be passed to the underlying design management system's check out command.

Value Returned

`t`	The files were checked out.
`nil`	An error occurred and the files were not checked out.

gdmdelete

```
gdmdelete(
     { G_gdmSpec | q_gdmSpecList }
     [ x_gdmOptions ]
     [ g_xtra ]
     )
     => t / nil
```

Description

Deletes the library, cell, view, directory, or file that *G_gdmSpec* represents from the workarea and the default configuration.

If *G_gdmSpec* represents a directory that is not a library element, this function works recursively on the directory only if the `GDM_RECURSE` option was set when *G_gdmSpec* was created with gdmCreateSpec.

This function is the SKILL equivalent of the `gdmdelete` command. See `gdmdelete` for more information.

Arguments

G_gdmSpec	The gdmSpec object of the library, cell, view, directory, or file to delete.
q_gdmSpecList	A gdmSpecList object containing gdmSpec objects.
x_gdmOptions	One of the following integers:
	`1`: Deletes files from the workarea only.
	`0`: Indicates no options are set. Default.
g_xtra	String containing additional arguments to be passed to the underlying design management system’s delete command.

Value Returned

`t`	The files were deleted.
`nil`	An error occurred and the files were not deleted.

gdmexport

```
gdmexport(
     { G_gdmSpec | q_gdmSpecList }
     g_destination
     [ g_version ]
     [ x_gdmOptions ]
     [ g_xtra ]
     )
     => t / nil
```

Description

Exports the library, cell, view, directory, or file that *G_gdmSpec* represents from the design management system data repository to the destination you specify.

If *G_gdmSpec* represents a directory that is not a library element, this function works recursively on the directory only if the `GDM_RECURSE` option was set when *G_gdmSpec* was created with gdmCreateSpec.

This function is the SKILL equivalent of the `gdmexport` command. See `gdmexport` for more information.

Arguments

`G_gdmSpec`	The gdmSpec object of the library, cell, view, directory, or file to export.
`q_gdmSpecList`	A gdmSpecList object containing gdmSpec objects.
`g_destination`	The location, a directory or file, to which to export the files. *`g_destination`* can be a file only if you export only one file.
`g_version`	The version of the files to export.
`x_gdmOptions`	One of the following integers: `1`: Exports all managed files in a view instead of only co-managed view files. `2`: Creates the directory tree structure of the source directory in the destination directory. `0`: Indicates no options are set. Default.
`g_xtra`	String containing additional arguments to be passed to the underlying design management system's export command.

Value Returned

`t`	The files were exported.
`nil`	An error occurred and the files were not exported.

gdmhistory

```
gdmhistory(
     { G_gdmSpec | q_gdmSpecList }
     x_information
     )
     => g_returnInfo / nil
```

Description

Returns information about the version history of a file.

This function is the SKILL equivalent of the `gdmhistory` command. See `gdmhistory` for more information.

Arguments

G_gdmSpec	The gdmSpec object of the file whose history you want to get.
q_gdmSpecList	A gdmSpecList object containing gdmSpec objects.
x_information	Specifies the kind of information to return. The following table lists the legal values for this argument.

Specify ...	**to get ...**
`1`	The file’s current version
`2`	Name of the file’s author
`3`	The date the file was created
`4`	The size of the file
`5`	The version from which the current version was created
`6`	The description that was submitted with the file
`7`	The design management status, such as checked out
`8`	The label/name attached on the version

Note: If you specify the *`x_information`* value as `0`, the function returns `nil`.

Value Returned

`g_returnInfo`	String containing the information requested.
`nil`	An error occurred and the information could not be returned.

gdmsetdefver

```
gdmsetdefver(
     { G_gdmSpec | q_gdmSpecList }
     g_version
     [ g_name ]
     [ x_gdmOptions ]
     [ g_xtra ]
     )
     => t / nil
```

Description

Sets `g_version` as the default version of the library, cell, view, directory, or file that `G_gdmSpec` represents.

This function is the SKILL equivalent of the `gdmsetdefver` command. See `gdmsetdefver` for more information.

Arguments

G_gdmSpec	The gdmSpec object of the library, cell, view, directory, or file whose default version you want to set.
q_gdmSpecList	A gdmSpecList object containing gdmSpec objects.
g_version	The version to set as the default version of the files; specified as a string.
g_name	The name, such as an RCS tag, that identifies a set of files. If you specify this argument, the default version is applied only to this set of files.
x_gdmOptions	This argument is currently ignored. The default is the integer `0`.
g_xtra	String containing additional arguments to be passed to the underlying design management system's set default version command.

Value Returned

`t`	The default version was set for the files.
`nil`	An error occurred and the default version was not set for the files.

gdmsetname

```
gdmsetname(
     { G_gdmSpec | q_gdmSpecList }
     g_name
     [ g_version ]
     [ x_gdmOptions ]
     [ g_xtra ]
     )
     => t / nil
```

Description

Associates the name you specify with the library, cell, view, directory, or file that *`G_gdmSpec`* represents.

If *`G_gdmSpec`* represents a directory that is not a library element, this function works recursively on the directory only if the `GDM_RECURSE` option was set when *`G_gdmSpec`* was created with gdmCreateSpec.

This function is the SKILL equivalent of the `gdmsetname` command. See `gdmsetname` for more information.

Arguments

`G_gdmSpec`	The library, cell, view, directory, or files to associate with *`g_name`*.
`q_gdmSpecList`	A gdmSpecList object containing gdmSpec objects.
`g_name`	The name to associate with the files. This is equivalent to an RCS tag name or a TDM release name.
`g_version`	The version of the files to associate with *`g_name`*. You can specify this argument only if *`G_gdmSpec`* represents a file or a view.
`x_gdmOptions`	This argument is currently ignored. The default is the integer `0`.
`g_xtra`	String containing additional arguments to be passed to the underlying design management system's set name command.

Value Returned

`t`	The name was associated with the files.

`nil`	An error occurred and the name was not associated with the files.

gdmstatus

```
gdmstatus(
     { G_gdmSpec | q_gdmSpecList }
     x_information
     )
     => l_fileStatus / nil
```

Description

Returns the design management status of the library, cell, view, directory, or file that *G_gdmSpec* represents.

This function is the SKILL equivalent of the `gdmstatus` command. See `gdmstatus` for more information.

Arguments

G_gdmSpec	The `gdmSpec` object of the library, cell, view, directory, or file.
q_gdmSpecList	A `gdmSpecList` object containing `gdmSpec` objects.
x_information	Specifies the kind of information to return. The following table lists the legal values for this argument.

Specify ...	**to get ...**
`1`	The `gdmSpec` object.
`2`	The library to which the file belongs.
`3`	The cell to which the file belongs.
`4`	The view to which the file belongs.
`5`	The path to the file, relative to the library to which it belongs or to the current directory.

Specify ...	to get ...
`6`	The design management status. Returns one of the following: ■ `gdmStateNone:` DMTYPE is set to a valid DM system. You may request GDM to attempt check in of an unmanaged state data into the named DM. This usually works within a managed library, depending upon correct DM setup and configuration including the cdsinfo.tag file. ■ `gdmStateCI:` File is managed and not checked out ■ `gdmStateCO`: File is checked out to current workarea ■ `gdmStateCOE`: File is checked out but not to current workarea ■ `gdmStateDel:` File is flagged as deleted ■ `gdmStateDir:` File is a directory ■ `gdmStateInactive:` File is inactive ■ `gdmStateUnmanageable:` DMTYPE is set to NONE. A GDM check in cannot be processed in the current configuration state. For example, a directory, which is unmanageable will not have its content expanded. Therefore, no subdirectory data will be returned for it, even if the GDM_RECURSE option is added. The UNMANAGEABLE state cannot process a GDM check-in within the current configuration state. It means the DMTYPE has been set to NONE by a cdsinfo.tag file.

Specify ...	**to get ...**
`7`	The status of the file in the workarea. Returns one of the following: ■ `gdmStateNone`: File is not in the workarea. ■ `gdmStateRead`: File is in the workarea but is read-only. ■ `gdmStateWrite`: File is checked out to current. workarea. ■ `gdmStateDir`: File is a directory. ■ `gdmStateErr`: File is in an inconsistent state.
`8`	Check whether the file has been modified in the workarea.
`9`	The version the file will be when it is checked in, or if it is already checked in, the current version.
`10`	The version of the file you will get if you check it out, or if it is already checked out, the version that was checked out.
`11`	The version of the file you will get if you update it.
`12`	The check-out location of the file.
`13`	The name of the person who checked out the file.
`14`	The name of the person who checked in the file.
`15`	The update needed status of the file. Returns one of the following: ■ `true`: Update is required. ■ `false`: Update is not required. ■ `unknown`: DM does not support this feature.

Value Returned

l_fileStatus	Returns a list of lists containing the status of each file in the cellview directory.
`nil`	An error occurred and the information could not be returned.

Example

```
strlist=gdmStatus(gdmCreateSpec("rodTrLib" "Design" "layout" master.tag" "CDBA")
15)
=> (("/rodTrLib/Design/layout/master.tag true"))
```

Returns `true`. The `master.tag` file needs to be updated in the <`rodTrLib | Design | layout`> cellview.

gdmsubmit

```
gdmsubmit(
     { G_gdmSpec | q_gdmSpecList }
     [ g_description ]
     [ g_name ]
     [ x_gdmOptions ]
     [ g_xtra ]
     )
     => t / nil
```

Description

Submits the files *G_gdmSpec* represents for the release. Co-managed files in a view are always submitted as a group; co-managed set behavior applies only if *G_gdmSpec* consists of library elements.

If *G_gdmSpec* represents a directory that is not a library element, this function works recursively on the directory only if the `GDM_RECURSE` option was set when *G_gdmSpec* was created with gdmCreateSpec.

This function is the SKILL equivalent of the `gdmsubmit` command. See `gdmsubmit` for more information.

Arguments

G_gdmSpec	The gdmSpec object of the library, cell, view, directory, or file to submit for the release.
q_gdmSpecList	A gdmSpecList object containing gdmSpec objects.
g_description	A description of the submit request; specified as a string.
g_name	The integration request name or release name; specified as a string.
x_gdmOptions	One of the following integers: `1`: Includes files that are checked out. `0`: Indicates no options are set. Default.
g_xtra	String containing additional arguments to be passed to the underlying design management system's submit command.

Value Returned

`t`	The files were submitted for the release.
`nil`	An error occurred and the files were not submitted for the release.

gdmupdate

```
gdmupdate(
     { G_gdmSpec | q_gdmSpecList }
     [ g_version ]
     [ g_name ]
     [ x_gdmOptions ]
     [ g_xtra ]
     )
     => t / nil
```

Description

Makes files in the workarea available for reading. Co-managed files are updated in the same grouping in which they were checked in; co-managed behavior applies only if *G_gdmSpec* consists of library elements.

If *G_gdmSpec* represents a directory that is not a library element, this function works recursively on the directory only if the `GDM_RECURSE` option was set when *G_gdmSpec* was created with gdmCreateSpec.

This function is the SKILL equivalent of the `gdmupdate` command. See `gdmupdate` for more information.

Arguments

`G_gdmSpec`	The gdmSpec object of the library, cell, view, directory, or file to update.
`q_gdmSpecList`	A gdmSpecList object containing gdmSpec objects.
`g_version`	The version of the files to update. If you specify this argument, do not specify *`g_name`*.
`g_name`	The name associated with the files; specified as a string. This is equivalent to an RCS tag name or a TDM release name. If you specify this argument, you must specify `nil` as the value of the *`g_version`* argument.
`x_gdmOptions`	One of the following integers: `1`: Forces an update even if there are modified files in the workarea. Modified files in the workarea are overwritten and any unmanaged files within the comanaged set are deleted. `0`: Indicates no options are set. Default.
`g_xtra`	String containing additional arguments to be passed to the underlying design management system's update command.

Value Returned

`t`	The files were updated.
`nil`	An error occurred and the files were not updated.

gdmExecute

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

Description

Command line executable.

Arguments

g_general	The name of the executable(s).

Value Returned

`t`	Command executed.
`nil`	Command not executed.

Example

```
gdmExecute( "crcsci" "crcs" )
```

gdmRemovename

```
gdmRemovename(
     G_gdmSpec
     [ g_version ]
     [ g_name ]

     )
     => t / nil
```

Description

Requests that GDM removes a name from the specified files.

If only the *name* argument is used, then the current files in the work area will have the name removed. If both *version* and *name* are used, then the specified version will have the name removed.

Arguments

G_gdmSpec	The `gdmSpec` object that you want to remove a name from.
g_version	The version that you want to remove the name from. NULL will apply to all versions.
`g_name`	The name to be removed from the version.

Value Returned

`t`	Name successfully removed.
`nil`	Command not executed.

Example

```
gdmRemovename ( spec "release10" )
```

gdmObjIsCreated

```
gdmObjIsCreated(
     g_name

     )
     => t / nil
```

Description

Checks whether a view or file has already been created by another user in the same DM.

Arguments

`g_name`	The view or file to examine.

Value Returned

`t`	The file or view has been created by a DM system.
`nil`	The file or view has not been created.

2

Name Mapping Functions

This chapter provides several SKILL functions for name spaces. These functions have the prefix `nmp`.

You can use `nmp` functions to

- Get a list of all current name spaces
- Check if an identifier is legal in a name space
- Map an identifier from one name space to another name space
- Map an identifier from a name space to an nmp Ident

  An nmp Ident is an opaque data structure that is the intermediate form to which identifiers are mapped when they are translated from one name space to another.
- Map an nmp Ident to a name space

This section describes the syntax and usage of nmp SKILL functions. For a complete list of nmp SKILL functions, see nmp SKILL Functions.

Prefixes Used in Argument Names

Argument names in the SKILL functions described in this section have a one-character prefix that denotes the data type of the argument. The prefix is followed by an underscore character (_). For example, *`t_identifier`* indicates the argument is a string, while *`l_namespaces`* indicates the argument is a list. The prefix and the underscore are just naming conventions; you do not need to type them when you specify the value of the argument.

The following data type prefixes are used in this section:

`l`	list
`t`	string

`m`	nmp Ident

nmpGetSpaceNames

```
nmpGetSpaceNames()
     => l_namespaces
```

Description

Returns a list of name spaces used by Cadence applications.

Arguments

This function has no arguments.

Value Returned

l_namespaces	List of name spaces used by Cadence applications.

nmpIsLegal*<NameSpace>*

```
nmpIsLegal<NameSpace> (
     t_identifier
     )
     => t / nil
```

Description

Checks if the identifier you specify is legal in the name space. In the function name, substitute *`<NameSpace>`* with any of the name spaces returned by the function `nmpGetSpaceNames()`. For example, `nmpIsLegalVerilog()` or `nmpIsLegalVHDL()`.

For a complete list of `nmpIsLegal` functions, see "nmpIsLegal*<NameSpace>* Functions" on page 66.

Argument

`t_identifier`	The identifier you want to check.

Value Returned

`t`	The identifier is legal in the name space.
`nil`	The identifier is illegal in the name space.

nmp*<NameSpace>*To*<NameSpace>*

```
nmp<NameSpace>To<NameSpace>(
     t_identifier
     )
     => t_identifier
```

Description

Maps `t_identifier` from the first name space to the second name space. In the function name, substitute the first `<NameSpace>` with the name space to which the identifier belongs and substitute the second `<NameSpace>` with the name space to which you want to convert the identifier. For example, `nmpCDBAToVerilog()` or `nmpVHDLAMSToVHDL()`.

You can get a list of Cadence name spaces from the `nmpGetSpaceNames()` function.

For a complete list of `nmp<NameSpace>To<NameSpace>` functions, see "nmp*<NameSpace>*To*<NameSpace>* Functions" on page 69.

Arguments

t_identifier	The identifier you want to map to another name space.

Value Returned

t_identifier	The identifier in the new name space.

nmp*<NameSpace>*ToIdent

```
nmp<NameSpace>ToIdent(
     t_identifier
     )
     => m_Ident
```

Description

Maps `t_identifier` from a name space to an nmp Ident. In the function name, substitute *<NameSpace>* with the name space to which the identifier belongs. For example, `nmpCDBAToIdent()`. You can get a list of Cadence name spaces from the `nmpGetSpaceNames()` function.

You can map the Ident that is returned from this function to another name space with the nmpIdentTo*<NameSpace>* function.

For a complete list of `nmp<NameSpace>ToIdent` functions, see "nmp*<NameSpace>*ToIdent Functions" on page 67.

Arguments

t_identifier	The identifier you want to map to Ident.

Value Returned

m_Ident	The identifier in Ident form. An nmp Ident is an opaque data structure that is the intermediate form to which identifiers are mapped when they are translated from one name space to another.

nmpIdentTo*<NameSpace>*

```
nmpIdentTo<NameSpace>(
     m_Ident
     )
     => t_identifier
```

Description

Maps an nmp Ident to a name space. In the function name, substitute `<NameSpace>` with the name space to which you want to map the Ident. For example, `nmpIdentToSpectre()`. You can get a list of Cadence name spaces from the `nmpGetSpaceNames()` function.

For a complete list of `nmpIdentTo<NameSpace>` functions, see "nmpIdentTo*<NameSpace>* Functions" on page 68.

Arguments

m_Ident	The Ident you want to map to the name space. An nmp Ident is an opaque data structure that is the intermediate form to which identifiers are mapped when they are translated from one name space to another.

Value Returned

t_identifier	The identifier in the name space you specified.

nmpPath*<NameSpace>***To***<NameSpace>*

```
nmpPath<NameSpace>To<NameSpace>(
     t_Inst
     )
     => t_Inst / nil
```

Description

Converts a simple instance vector from the first name space to the second name space.

For a complete list of `nmpPath<NameSpace>To<NameSpace>` functions, see "nmpPath*<NameSpace>*To*<NameSpace>* Functions" on page 83.

Argument

t_Inst	A simple (non-hierarchical) instance vector you want to convert to another name space.

Value Returned

t_Inst	The instance vector in the new name space.
`nil`	The input string is not legal in the first name space.

nmp SKILL Functions

This section lists all nmp SKILL functions.

- nmpIsLegal*<NameSpace> Functions*
- nmp*<NameSpace>ToIdent Functions*
- nmpIdentTo*<NameSpace> Functions*
- nmp*<NameSpace>To<NameSpace> Functions*
- nmpPath*<NameSpace>To<NameSpace> Functions*

nmpIsLegal*<NameSpace>* Functions

The following table lists all current `nmpIsLegal`*<NameSpace>* functions. For information about these functions, see nmpIsLegal*<NameSpace>*.

`nmpIsLegalCDBA()`	Checks if the identifier you specify is legal in CDBA.
`nmpIsLegalConcept()`	Checks if the identifier you specify is legal in Concept.
`nmpIsLegalDef()`	Checks if the identifier you specify is legal in DEF.
`nmpIsLegalGcf()`	Checks if the identifier you specify is legal in GCF.
`nmpIsLegalGenesis()`	Checks if the identifier you specify is legal in Genesis.
`nmpIsLegalLef()`	Checks if the identifier you specify is legal in LEF.
`nmpIsLegalPrint()`	Checks if the identifier you specify is legal in Print.
`nmpIsLegalSdf()`	Checks if the identifier you specify is legal in SDF.
`nmpIsLegalSpf()`	Checks if the identifier you specify is legal in SPF.
`nmpIsLegalSpectre()`	Checks if the identifier you specify is legal in Spectre.
`nmpIsLegalSpectreHDL()`	Checks if the identifier you specify is legal in SpectreHDL.
`nmpIsLegalSpef()`	Checks if the identifier you specify is legal in SPEF.
`nmpIsLegalSpice()`	Checks if the identifier you specify is legal in Spice.
`nmpIsLegalSysVerilog()`	Checks if the identifier you specify is legal in SysVerilog.
`nmpIsLegalVerilog()`	Checks if the identifier you specify is legal in Verilog.
`nmpIsLegalVerilogA()`	Checks if the identifier you specify is legal in VerilogA.
`nmpIsLegalVerilogAMS()`	Checks if the identifier you specify is legal in VerilogAMS.
`nmpIsLegalVHDL()`	Checks if the identifier you specify is legal in VHDL.
`nmpIsLegalVHDLAMS()`	Checks if the identifier you specify is legal in VHDLAMS.
`nmpIsLegalVHDL87()`	Checks if the identifier you specify is legal in VHDL87.

nmp`<NameSpace>`**ToIdent Functions**

The following table lists all current `nmp<NameSpace>ToIdent` functions. For information about these functions, see nmp<*NameSpace*>*ToIdent*.

`nmpCDBAToIdent()`	Maps identifier from CDBA to an nmp Ident.
`nmpConceptToIdent()`	Maps identifier from Concept to an nmp Ident.
`nmpDefToIdent()`	Maps identifier from Def to an nmp Ident.
`nmpGcfToIdent()`	Maps identifier from Gcf to an nmp Ident.
`nmpGenesisToIdent()`	Maps identifier from Genesis to an nmp Ident.
`nmpLefToIdent()`	Maps identifier from Lef to an nmp Ident.
`nmpPrintToIdent()`	Maps identifier from Print to an nmp Ident.
`nmpSdfToIdent()`	Maps identifier from Sdf to an nmp Ident.
`nmpSpfToIdent()`	Maps identifier from Spf to an nmp Ident.
`nmpSpectreToIdent()`	Maps identifier from Spectre to an nmp Ident.
`nmpSpectreHDLToIdent()`	Maps identifier from SpectreHDL to an nmp Ident.
`nmpSpefToIdent()`	Maps identifier from Spef to an nmp Ident.
`nmpSpiceToIdent()`	Maps identifier from Spice to an nmp Ident.
`nmpSysVerilogToIdent()`	Maps identifier from SysVerilog to an nmp Ident.
`nmpVerilogToIdent()`	Maps identifier from Verilog to an nmp Ident.
`nmpVerilogAToIdent()`	Maps identifier from VerilogA to an nmp Ident.
`nmpVerilogAMSToIdent()`	Maps identifier from VerilogAMS to an nmp Ident.
`nmpVHDLToIdent()`	Maps identifier from VHDL to an nmp Ident.
`nmpVHDLAMSToIdent()`	Maps identifier from VHDLAMS to an nmp Ident.
`nmpVHDL87ToIdent()`	Maps identifier from VHDL87 to an nmp Ident.

nmpIdentTo*<NameSpace>* Functions

The following table lists all current `nmpIdentTo`*`<NameSpace>`* functions. For information about these functions, see nmpIdentTo*<NameSpace>*.

`nmpIdentToCDBA()`	Maps an nmp Ident to CDBA.
`nmpIdentToConcept()`	Maps an nmp Ident to Concept.
`nmpIdentToDef()`	Maps an nmp Ident to Def.
`nmpIdentToGcf()`	Maps an nmp Ident to Gcf.
`nmpIdentToGenesis()`	Maps an nmp Ident to Genesis.
`nmpIdentToLef()`	Maps an nmp Ident to Lef.
`nmpIdentToPrint()`	Maps an nmp Ident to Print.
`nmpIdentToSdf()`	Maps an nmp Ident to Sdf.
`nmpIdentToSpf()`	Maps an nmp Ident to Spf.
`nmpIdentToSpectre()`	Maps an nmp Ident to Spectre.
`nmpIdentToSpectreHDL()`	Maps an nmp Ident to SpectreHDL.
`nmpIdentToSpef()`	Maps an nmp Ident to Spef.
`nmpIdentToSpice()`	Maps an nmp Ident to Spice.
`nmpIdentToSysVerilog()`	Maps an nmp Ident to SysVerilog.
`nmpIdentToVerilog()`	Maps an nmp Ident to Verilog.
`nmpIdentToVerilogA()`	Maps an nmp Ident to VerilogA.
`nmpIdentToVerilogAMS()`	Maps an nmp Ident to VerilogAMS.
`nmpIdentToVHDL()`	Maps an nmp Ident to VHDL.
`nmpIdentToVHDLAMS()`	Maps an nmp Ident to VHDLAMS.
`nmpIdentToVHDL87()`	Maps an nmp Ident to VHDL87.

nmp*<NameSpace>*To*<NameSpace>* Functions

The following table lists all current `nmp`*`<NameSpace>`*`To`*`<NameSpace>`* functions. For information about these functions, see nmp*<NameSpace>To<NameSpace>*.

`nmpAsciiToSysVerilog()`	Maps identifier from Ascii to SysVerilog.
`nmpCDBAToConcept()`	Maps identifier from CDBA to Concept.
`nmpCDBAToCDBAFlat()`	Maps identifier from CDBA to CDBAFlat.
`nmpCDBAToDef()`	Maps identifier from CDBA to Def.
`nmpCDBAToGcf()`	Maps identifier from CDBA to Gcf.
`nmpCDBAToGenesis()`	Maps identifier from CDBA to Genesis.
`nmpCDBAToLef()`	Maps identifier from CDBA to Lef.
`nmpCDBAToPrint()`	Maps identifier from CDBA to Print.
`nmpCDBAToSdf()`	Maps identifier from CDBA to Sdf.
`nmpCDBAToSpf()`	Maps identifier from CDBA to Spf.
`nmpCDBAToSpectre()`	Maps identifier from CDBA to Spectre.
`nmpCDBAToSpectreHDL()`	Maps identifier from CDBA to SpectreHDL.
`nmpCDBAToSpef()`	Maps identifier from CDBA to Spef.
`nmpCDBAToSpice()`	Maps identifier from CDBA to Spice.
`nmpCDBAToSysVerilog()`	Maps identifier from CDBA to SysVerilog.
`nmpCDBAToVerilog()`	Maps identifier from CDBA to Verilog.
`nmpCDBAToVerilogA()`	Maps identifier from CDBA to VerilogA.
`nmpCDBAToVerilogAMS()`	Maps identifier from CDBA to VerilogAMS.
`nmpCDBAToVHDL()`	Maps identifier from CDBA to VHDL.
`nmpCDBAToVHDLAMS()`	Maps identifier from CDBA to VHDLAMS.
`nmpCDBAToVHDL87()`	Maps identifier from CDBA to VHDL87.
`nmpCDBAFlatToCDBA()`	Maps identifier from CDBAFlat to CDBA.
`nmpConceptToCDBA()`	Maps identifier from Concept to CDBA.
`nmpConceptToDef()`	Maps identifier from Concept to Def.
`nmpConceptToGcf()`	Maps identifier from Concept to Gcf.

`nmpConceptToGenesis()`	Maps identifier from Concept to Genesis.
`nmpConceptToLef()`	Maps identifier from Concept to Lef.
`nmpConceptToPrint()`	Maps identifier from Concept to Print.
`nmpConceptToSdf()`	Maps identifier from Concept to Sdf.
`nmpConceptToSpf()`	Maps identifier from Concept to Spf.
`nmpConceptToSpectre()`	Maps identifier from Concept to Spectre.
`nmpConceptToSpectreHDL()`	Maps identifier from Concept to SpectreHDL.
`nmpConceptToSpef()`	Maps identifier from Concept to Spef.
`nmpConceptToSpice()`	Maps identifier from Concept to Spice.
`nmpConceptToSysVerilog()`	Maps identifier from Concept to SysVerilog.
`nmpConceptToVerilog()`	Maps identifier from Concept to Verilog.
`nmpConceptToVerilogA()`	Maps identifier from Concept to VerilogA.
`nmpConceptToVerilogAMS()`	Maps identifier from Concept to VerilogAMS.
`nmpConceptToVHDL()`	Maps identifier from Concept to VHDL.
`nmpConceptToVHDLAMS()`	Maps identifier from Concept to VHDLAMS.
`nmpConceptToVHDL87()`	Maps identifier from Concept to VHDL87.
`nmpDefToCDBA()`	Maps identifier from Def to CDBA.
`nmpDefToConcept()`	Maps identifier from Def to Concept.
`nmpDefToGcf()`	Maps identifier from Def to Gcf.
`nmpDefToGenesis()`	Maps identifier from Def to Genesis.
`nmpDefToLef()`	Maps identifier from Def to Lef.
`nmpDefToPrint()`	Maps identifier from Def to Print.
`nmpDefToSdf()`	Maps identifier from Def to Sdf.
`nmpDefToSpf()`	Maps identifier from Def to Spf.
`nmpDefToSpectre()`	Maps identifier from Def to Spectre.
`nmpDefToSpectreHDL()`	Maps identifier from Def to SpectreHDL.
`nmpDefToSpef()`	Maps identifier from Def to Spef.
`nmpDefToSpice()`	Maps identifier from Def to Spice.
`nmpDefToSysVerilog()`	Maps identifier from Def to SysVerilog.

`nmpDefToVerilog()`	Maps identifier from Def to Verilog.
`nmpDefToVerilogA()`	Maps identifier from Def to VerilogA.
`nmpDefToVerilogAMS()`	Maps identifier from Def to VerilogAMS.
`nmpDefToVHDL()`	Maps identifier from Def to VHDL.
`nmpDefToVHDLAMS()`	Maps identifier from Def to VHDLAMS.
`nmpDefToVHDL87()`	Maps identifier from Def to VHDL87.
`nmpGcfToCDBA()`	Maps identifier from Gcf to CDBA.
`nmpGcfToConcept()`	Maps identifier from Gcf to Concept.
`nmpGcfToDef()`	Maps identifier from Gcf to Def.
`nmpGcfToGenesis()`	Maps identifier from Gcf to Genesis.
`nmpGcfToLef()`	Maps identifier from Gcf to Lef.
`nmpGcfToPrint()`	Maps identifier from Gcf to Print.
`nmpGcfToSdf()`	Maps identifier from Gcf to Sdf.
`nmpGcfToSpf()`	Maps identifier from Gcf to Spf.
`nmpGcfToSpectre()`	Maps identifier from Gcf to Spectre.
`nmpGcfToSpectreHDL()`	Maps identifier from Gcf to SpectreHDL.
`nmpGcfToSpef()`	Maps identifier from Gcf to Spef.
`nmpGcfToSpice()`	Maps identifier from Gcf to Spice.
`nmpGcfToSysVerilog()`	Maps identifier from Gcf to SysVerilog.
`nmpGcfToVerilog()`	Maps identifier from Gcf to Verilog.
`nmpGcfToVerilogA()`	Maps identifier from Gcf to VerilogA.
`nmpGcfToVerilogAMS()`	Maps identifier from Gcf to VerilogAMS.
`nmpGcfToVHDL()`	Maps identifier from Gcf to VHDL.
`nmpGcfToVHDLAMS()`	Maps identifier from Gcf to VHDLAMS.
`nmpGcfToVHDL87()`	Maps identifier from Gcf to VHDL87.
`nmpGenesisToCDBA()`	Maps identifier from Genesis to CDBA.
`nmpGenesisToConcept()`	Maps identifier from Genesis to Concept.
`nmpGenesisToDef()`	Maps identifier from Genesis to Def.
`nmpGenesisToGcf()`	Maps identifier from Genesis to Gcf.

`nmpGenesisToLef()`	Maps identifier from Genesis to Lef.
`nmpGenesisToPrint()`	Maps identifier from Genesis to Print.
`nmpGenesisToSdf()`	Maps identifier from Genesis to Sdf.
`nmpGenesisToSpf()`	Maps identifier from Genesis to Spf.
`nmpGenesisToSpectre()`	Maps identifier from Genesis to Spectre.
`nmpGenesisToSpectreHDL()`	Maps identifier from Genesis to SpectreHDL.
`nmpGenesisToSpef()`	Maps identifier from Genesis to Spef.
`nmpGenesisToSpice()`	Maps identifier from Genesis to Spice.
`nmpGenesisToSysVerilog()`	Maps identifier from Genesis to SysVerilog.
`nmpGenesisToVerilog()`	Maps identifier from Genesis to Verilog.
`nmpGenesisToVerilogA()`	Maps identifier from Genesis to VerilogA.
`nmpGenesisToVerilogAMS()`	Maps identifier from Genesis to VerilogAMS.
`nmpGenesisToVHDL()`	Maps identifier from Genesis to VHDL.
`nmpGenesisToVHDLAMS()`	Maps identifier from Genesis to VHDLAMS.
`nmpGenesisToVHDL87()`	Maps identifier from Genesis to VHDL87.
`nmpLefToCDBA()`	Maps identifier from Lef to CDBA.
`nmpLefToConcept()`	Maps identifier from Lef to Concept.
`nmpLefToDef()`	Maps identifier from Lef to Def.
`nmpLefToGcf()`	Maps identifier from Lef to Gcf.
`nmpLefToGenesis()`	Maps identifier from Lef to Genesis.
`nmpLefToPrint()`	Maps identifier from Lef to Print.
`nmpLefToSdf()`	Maps identifier from Lef to Sdf.
`nmpLefToSpf()`	Maps identifier from Lef to Spf.
`nmpLefToSpectre()`	Maps identifier from Lef to Spectre.
`nmpLefToSpectreHDL()`	Maps identifier from Lef to SpectreHDL.
`nmpLefToSpef()`	Maps identifier from Lef to Spef.
`nmpLefToSpice()`	Maps identifier from Lef to Spice.
`nmpLefToSysVerilog()`	Maps identifier from Lef to SysVerilog.
`nmpLefToVerilog()`	Maps identifier from Lef to Verilog.

`nmpLefToVerilogA()`	Maps identifier from Lef to VerilogA.
`nmpLefToVerilogAMS()`	Maps identifier from Lef to VerilogAMS.
`nmpLefToVHDL()`	Maps identifier from Lef to VHDL.
`nmpLefToVHDLAMS()`	Maps identifier from Lef to VHDLAMS.
`nmpLefToVHDL87()`	Maps identifier from Lef to VHDL87.
`nmpPrintToCDBA()`	Maps identifier from Print to CDBA.
`nmpPrintToConcept()`	Maps identifier from Print to Concept.
`nmpPrintToDef()`	Maps identifier from Print to Def.
`nmpPrintToGcf()`	Maps identifier from Print to Gcf.
`nmpPrintToGenesis()`	Maps identifier from Print to Genesis.
`nmpPrintToLef()`	Maps identifier from Print to Lef.
`nmpPrintToSdf()`	Maps identifier from Print to Sdf.
`nmpPrintToSpf()`	Maps identifier from Print to Spf.
`nmpPrintToSpectre()`	Maps identifier from Print to Spectre.
`nmpPrintToSpectreHDL()`	Maps identifier from Print to SpectreHDL.
`nmpPrintToSpef()`	Maps identifier from Print to Spef.
`nmpPrintToSpice()`	Maps identifier from Print to Spice.
`nmpPrintToSysVerilog()`	Maps identifier from Print to SysVerilog.
`nmpPrintToVerilog()`	Maps identifier from Print to Verilog.
`nmpPrintToVerilogA()`	Maps identifier from Print to VerilogA.
`nmpPrintToVerilogAMS()`	Maps identifier from Print to VerilogAMS.
`nmpPrintToVHDL()`	Maps identifier from Print to VHDL.
`nmpPrintToVHDLAMS()`	Maps identifier from Print to VHDLAMS.
`nmpPrintToVHDL87()`	Maps identifier from Print to VHDL87.
`nmpSdfToCDBA()`	Maps identifier from Sdf to CDBA.
`nmpSdfToConcept()`	Maps identifier from Sdf to Concept.
`nmpSdfToDef()`	Maps identifier from Sdf to Def.
`nmpSdfToGcf()`	Maps identifier from Sdf to Gcf.
`nmpSdfToGenesis()`	Maps identifier from Sdf to Genesis.

`nmpSdfToLef()`	Maps identifier from Sdf to Lef.
`nmpSdfToPrint()`	Maps identifier from Sdf to Print.
`nmpSdfToSpf()`	Maps identifier from Sdf to Spf.
`nmpSdfToSpectre()`	Maps identifier from Sdf to Spectre.
`nmpSdfToSpectreHDL()`	Maps identifier from Sdf to SpectreHDL.
`nmpSdfToSpef()`	Maps identifier from Sdf to Spef.
`nmpSdfToSpice()`	Maps identifier from Sdf to Spice.
`nmpSdfToSysVerilog()`	Maps identifier from Sdf to SysVerilog.
`nmpSdfToVerilog()`	Maps identifier from Sdf to Verilog.
`nmpSdfToVerilogA()`	Maps identifier from Sdf to VerilogA.
`nmpSdfToVerilogAMS()`	Maps identifier from Sdf to VerilogA.
`nmpSdfToVHDL()`	Maps identifier from Sdf to VHDL.
`nmpSdfToVHDLAMS()`	Maps identifier from Sdf to VerilogAMS.
`nmpSdfToVHDL87()`	Maps identifier from Sdf to VHDL87.
`nmpSpfToCDBA()`	Maps identifier from Spf to CDBA.
`nmpSpfToConcept()`	Maps identifier from Spf to Concept.
`nmpSpfToDef()`	Maps identifier from Spf to Def.
`nmpSpfToGcf()`	Maps identifier from Spf to Gcf.
`nmpSpfToGenesis()`	Maps identifier from Spf to Genesis.
`nmpSpfToLef()`	Maps identifier from Spf to Lef.
`nmpSpfToPrint()`	Maps identifier from Spf to Print.
`nmpSpfToSdf()`	Maps identifier from Spf to Sdf.
`nmpSpfToSpectre()`	Maps identifier from Spf to Spectre.
`nmpSpfToSpectreHDL()`	Maps identifier from Spf to SpectreHDL.
`nmpSpfToSpef()`	Maps identifier from Spf to Spef.
`nmpSpfToSpice()`	Maps identifier from Spf to Spice.
`nmpSpfToSysVerilog()`	Maps identifier from Spf to SysVerilog.
`nmpSpfToVerilog()`	Maps identifier from Spf to Verilog.
`nmpSpfToVerilogA()`	Maps identifier from Spf to VerilogA.

`nmpSpfToVerilogAMS()`	Maps identifier from Spf to VerilogAMS.
`nmpSpfToVHDL()`	Maps identifier from Spf to VHDL.
`nmpSpfToVHDLAMS()`	Maps identifier from Spf to VHDLAMS.
`nmpSpfToVHDL87()`	Maps identifier from Spf to VHDL87.
`nmpSpectreToCDBA()`	Maps identifier from Spectre to CDBA.
`nmpSpectreToConcept()`	Maps identifier from Spectre to Concept.
`nmpSpectreToDef()`	Maps identifier from Spectre to Def.
`nmpSpectreToGcf()`	Maps identifier from Spectre to Gcf.
`nmpSpectreToGenesis()`	Maps identifier from Spectre to Genesis.
`nmpSpectreToLef()`	Maps identifier from Spectre to Lef.
`nmpSpectreToPrint()`	Maps identifier from Spectre to Print
`nmpSpectreToSdf()`	Maps identifier from Spectre to Sdf.
`nmpSpectreToSpf()`	Maps identifier from Spectre to Spf.
`nmpSpectreToSpectreHDL()`	Maps identifier from Spectre to SpectreHDL.
`nmpSpectreToSpef()`	Maps identifier from Spectre to Spef.
`nmpSpectreToSpice()`	Maps identifier from Spectre to Spice.
`nmpSpectreToSysVerilog()`	Maps identifier from Spectre to SysVerilog.
`nmpSpectreToVerilog()`	Maps identifier from Spectre to Verilog.
`nmpSpectreToVerilogA()`	Maps identifier from Spectre to VerilogA.
`nmpSpectreToVerilogAMS()`	Maps identifier from Spectre to VerilogAMS.
`nmpSpectreToVHDL()`	Maps identifier from Spectre to VHDL.
`nmpSpectreToVHDLAMS()`	Maps identifier from Spectre to VHDLAMS.
`nmpSpectreToVHDL87()`	Maps identifier from Spectre to VHDL87.
`nmpSpectreHDLToCDBA()`	Maps identifier from SpectreHDL to CDBA.
`nmpSpectreHDLToConcept()`	Maps identifier from SpectreHDL to Concept.
`nmpSpectreHDLToDef()`	Maps identifier from SpectreHDL to Def.
`nmpSpectreHDLToGcf()`	Maps identifier from SpectreHDL to Gcf.
`nmpSpectreHDLToGenesis()`	Maps identifier from SpectreHDL to Genesis.
`nmpSpectreHDLToLef()`	Maps identifier from SpectreHDL to Lef.

`nmpSpectreHDLToPrint()`	Maps identifier from SpectreHDL to Print.
`nmpSpectreHDLToSdf()`	Maps identifier from SpectreHDL to Sdf.
`nmpSpectreHDLToSpf()`	Maps identifier from SpectreHDL to Spf.
`nmpSpectreHDLToSpectre()`	Maps identifier from SpectreHDL to Spectre.
`nmpSpectreHDLToSpef()`	Maps identifier from SpectreHDL to Spef.
`nmpSpectreHDLToSpice()`	Maps identifier from SpectreHDL to Spice.
`nmpSpectreHDLToSysVerilog()`	Maps identifier from Spectre to SysVerilog.
`nmpSpectreHDLToVerilog()`	Maps identifier from SpectreHDL to Verilog.
`nmpSpectreHDLToVerilogA()`	Maps identifier from SpectreHDL to VerilogA.
`nmpSpectreHDLToVerilogAMS()`	Maps identifier from SpectreHDL to VerilogAMS.
`nmpSpectreHDLToVHDL()`	Maps identifier from SpectreHDL to VHDL.
`nmpSpectreHDLToVHDLAMS()`	Maps identifier from SpectreHDL to VHDLAMS.
`nmpSpectreHDLToVHDL87()`	Maps identifier from SpectreHDL to VHDL87.
`nmpSpefToCDBA()`	Maps identifier from Spef to CDBA.
`nmpSpefToConcept()`	Maps identifier from Spef to Concept.
`nmpSpefToDef()`	Maps identifier from Spef to Def.
`nmpSpefToGcf()`	Maps identifier from Spef to Gcf.
`nmpSpefToGenesis()`	Maps identifier from Spef to Genesis.
`nmpSpefToLef()`	Maps identifier from Spef to Lef.
`nmpSpefToPrint()`	Maps identifier from Spef to Print.
`nmpSpefToSdf()`	Maps identifier from Spef to Sdf.
`nmpSpefToSpf()`	Maps identifier from Spef to Spf.
`nmpSpefToSpectre()`	Maps identifier from Spef to Spectre.
`nmpSpefToSpectreHDL()`	Maps identifier from Spef to SpectreHDL.
`nmpSpefToSpice()`	Maps identifier from Spef to Spice.
`nmpSpefToSysVerilog()`	Maps identifier from Spef to SysVerilog.
`nmpSpefToVerilog()`	Maps identifier from Spef to Verilog.
`nmpSpefToVerilogA()`	Maps identifier from Spef to VerilogA.
`nmpSpefToVerilogAMS()`	Maps identifier from Spef to VerilogAMS.

`nmpSpefToVHDL()`	Maps identifier from Spef to VHDL.
`nmpSpefToVHDLAMS()`	Maps identifier from Spef to VHDLAMS.
`nmpSpefToVHDL87()`	Maps identifier from Spef to VHDL87.
`nmpSpiceToCDBA()`	Maps identifier from Spice to CDBA.
`nmpSpiceToConcept()`	Maps identifier from Spice to Concept.
`nmpSpiceToDef()`	Maps identifier from Spice to Def.
`nmpSpiceToGcf()`	Maps identifier from Spice to Gcf.
`nmpSpiceToGenesis()`	Maps identifier from Spice to Genesis.
`nmpSpiceToLef()`	Maps identifier from Spice to Lef.
`nmpSpiceToPrint()`	Maps identifier from Spice to Print.
`nmpSpiceToSdf()`	Maps identifier from Spice to Sdf.
`nmpSpiceToSpf()`	Maps identifier from Spice to Spf.
`nmpSpiceToSpectre()`	Maps identifier from Spice to Spectre.
`nmpSpiceToSpectreHDL()`	Maps identifier from Spice to SpectreHDL.
`nmpSpiceToSpef()`	Maps identifier from Spice to Spef.
`nmpSpiceToSysVerilog()`	Maps identifier from Spice to SysVerilog.
`nmpSpiceToVerilog()`	Maps identifier from Spice to Verilog.
`nmpSpiceToVerilogA()`	Maps identifier from Spice to VerilogA.
`nmpSpiceToVerilogAMS()`	Maps identifier from Spice to VerilogAMS.
`nmpSpiceToVHDL()`	Maps identifier from Spice to VHDL.
`nmpSpiceToVHDLAMS()`	Maps identifier from Spice to VHDLAMS.
`nmpSpiceToVHDL87()`	Maps identifier from Spice to VHDL87.
`nmpSysVerilogToAscii()`	Maps identifier from SysVerilog to Ascii.
`nmpSysVerilogToCDBA()`	Maps identifier from SysVerilog to CDBA.
`nmpSysVerilogToConcept()`	Maps identifier from SysVerilog to Concept.
`nmpSysVerilogToDef()`	Maps identifier from SysVerilog to Def.
`nmpSysVerilogToGcf()`	Maps identifier from SysVerilog to Gcf.
`nmpSysVerilogToGenesis()`	Maps identifier from SysVerilog to Genesis.
`nmpSysVerilogToLef()`	Maps identifier from SysVerilog to Lef.

`nmpSysVerilogToPrint()`	Maps identifier from SysVerilog to Print.
`nmpSysVerilogToSdf()`	Maps identifier from SysVerilog to Sdf.
`nmpSysVerilogToSpf()`	Maps identifier from SysVerilog to Spf.
`nmpSysVerilogToSpectre()`	Maps identifier from SysVerilog to Spectre.
`nmpSysVerilogToSpectreHDL()`	Maps identifier from SysVerilog to SpectreHDL.
`nmpSysVerilogToSpef()`	Maps identifier from SysVerilog to Spef.
`nmpSysVerilogToSpice()`	Maps identifier from SysVerilog to Spice.
`nmpSysVerilogToVHDL()`	Maps identifier from SysVerilog to VHDL.
`nmpSysVerilogToVHDL87()`	Maps identifier from SysVerilog to VHDL87.
`nmpSysVerilogToVHDLAMS()`	Maps identifier from SysVerilog to VHDLAMS.
`nmpSysVerilogToVerilog()`	Maps identifier from SysVerilog to Verilog.
`nmpSysVerilogToVerilogA()`	Maps identifier from SysVerilog to VerilogA.
`nmpSysVerilogToVerilogAMS()`	Maps identifier from SysVerilog to VerilogAMS.
`nmpVerilogToCDBA()`	Maps identifier from Verilog to CDBA.
`nmpVerilogToConcept()`	Maps identifier from Verilog to Concept.
`nmpVerilogToDef()`	Maps identifier from Verilog to Def.
`nmpVerilogToGcf()`	Maps identifier from Verilog to Gcf.
`nmpVerilogToGenesis()`	Maps identifier from Verilog to Genesis.
`nmpVerilogToLef()`	Maps identifier from Verilog to Lef.
`nmpVerilogToPrint()`	Maps identifier from Verilog to Print.
`nmpVerilogToSdf()`	Maps identifier from Verilog to Sdf.
`nmpVerilogToSpf()`	Maps identifier from Verilog to Spf.
`nmpVerilogToSpectre()`	Maps identifier from Verilog to Spectre.
`nmpVerilogToSpectreHDL()`	Maps identifier from Verilog to SpectreHDL.
`nmpVerilogToSpef()`	Maps identifier from Verilog to Spef.
`nmpVerilogToSpice()`	Maps identifier from Verilog to Spice.
`nmpVerilogToSysVerilog()`	Maps identifier from Verilog to SysVerilog.
`nmpVerilogToVerilogA()`	Maps identifier from Verilog to VerilogA.
`nmpVerilogToVerilogAMS()`	Maps identifier from Verilog to VerilogAMS.

`nmpVerilogToVHDL()`	Maps identifier from Verilog to VHDL.
`nmpVerilogToVHDLAMS()`	Maps identifier from Verilog to VHDLAMS.
`nmpVerilogToVHDL87()`	Maps identifier from Verilog to VHDL87.
`nmpVerilogAToCDBA()`	Maps identifier from VerilogA to CDBA.
`nmpVerilogAToConcept()`	Maps identifier from VerilogA to Concept.
`nmpVerilogAToDef()`	Maps identifier from VerilogA to Def.
`nmpVerilogAToGcf()`	Maps identifier from VerilogA to Gcf.
`nmpVerilogAToGenesis()`	Maps identifier from VerilogA to Genesis.
`nmpVerilogAToLef()`	Maps identifier from VerilogA to Lef.
`nmpVerilogAToPrint()`	Maps identifier from VerilogA to Print.
`nmpVerilogAToSdf()`	Maps identifier from VerilogA to Sdf.
`nmpVerilogAToSpf()`	Maps identifier from VerilogA to Spf.
`nmpVerilogAToSpectre()`	Maps identifier from VerilogA to Spectre.
`nmpVerilogAToSpectreHDL()`	Maps identifier from VerilogA to SpectreHDL.
`nmpVerilogAToSpef()`	Maps identifier from VerilogA to Spef.
`nmpVerilogAToSpice()`	Maps identifier from VerilogA to Spice.
`nmpVerilogAToSysVerilog()`	Maps identifier from VerilogA to SysVerilog.
`nmpVerilogAToVerilog()`	Maps identifier from VerilogA to Verilog.
`nmpVerilogAToVerilogAMS()`	Maps identifier from VerilogA to VerilogAMS.
`nmpVerilogAToVHDL()`	Maps identifier from VerilogA to VHDL.
`nmpVerilogAToVHDLAMS()`	Maps identifier from VerilogA to VHDLAMS.
`nmpVerilogAToVHDL87()`	Maps identifier from VerilogA to VHDL87.
`nmpVerilogAMSToCDBA()`	Maps identifier from VerilogAMS to CDBA.
`nmpVerilogAMSToConcept()`	Maps identifier from VerilogAMS to Concept.
`nmpVerilogAMSToDef()`	Maps identifier from VerilogAMS to Def.
`nmpVerilogAMSToGcf()`	Maps identifier from VerilogAMS to Gcf.
`nmpVerilogAMSToGenesis()`	Maps identifier from VerilogAMS to Genesis.
`nmpVerilogAMSToLef()`	Maps identifier from VerilogAMS to Lef.
`nmpVerilogAMSToPrint()`	Maps identifier from VerilogAMS to Print.

`nmpVerilogAMSToSdf()`	Maps identifier from VerilogAMS to Sdf.
`nmpVerilogAMSToSpf()`	Maps identifier from VerilogAMS to Spf.
`nmpVerilogAMSToSpectre()`	Maps identifier from VerilogAMS to Spectre.
`nmpVerilogAMSToSpectreHDL()`	Maps identifier from VerilogAMS to SpectreHDL.
`nmpVerilogAMSToSpef()`	Maps identifier from VerilogAMS to Spef.
`nmpVerilogAMSToSpice()`	Maps identifier from VerilogAMS to Spice.
`nmpVerilogAMSToSysVerilog()`	Maps identifier from VerilogAMS to SysVerilog.
`nmpVerilogAMSToVerilog()`	Maps identifier from VerilogAMS to Verilog.
`nmpVerilogAMSToVerilogA()`	Maps identifier from VerilogAMS to VerilogA.
`nmpVerilogAMSToVHDL()`	Maps identifier from VerilogAMS to VHDL.
`nmpVerilogAMSToVHDLAMS()`	Maps identifier from VerilogAMS to VHDLAMS.
`nmpVerilogAMSToVHDL87()`	Maps identifier from VerilogAMS to VHDL87.
`nmpVHDLToCDBA()`	Maps identifier from VHDL to CDBA.
`nmpVHDLToConcept()`	Maps identifier from VHDL to Concept.
`nmpVHDLToDef()`	Maps identifier from VHDL to Def.
`nmpVHDLToGcf()`	Maps identifier from VHDL to Gcf.
`nmpVHDLToGenesis()`	Maps identifier from VHDL to Genesis.
`nmpVHDLToLef()`	Maps identifier from VHDL to Lef.
`nmpVHDLToPrint()`	Maps identifier from VHDL to Print.
`nmpVHDLToSdf()`	Maps identifier from VHDL to Sdf.
`nmpVHDLToSpf()`	Maps identifier from VHDL to Spf.
`nmpVHDLToSpectre()`	Maps identifier from VHDL to Spectre.
`nmpVHDLToSpectreHDL()`	Maps identifier from VHDL to SpectreHDL.
`nmpVHDLToSpef()`	Maps identifier from VHDL to Spef.
`nmpVHDLToSpice()`	Maps identifier from VHDL to Spice.
`nmpVHDLToSysVerilog()`	Maps identifier from VHDL to SysVerilog.
`nmpVHDLToVerilog()`	Maps identifier from VHDL to Verilog.
`nmpVHDLToVerilogA()`	Maps identifier from VHDL to VerilogA.
`nmpVHDLToVerilogAMS()`	Maps identifier from VHDL to VerilogAMS.

`nmpVHDLToVHDLAMS()`	Maps identifier from VHDL to VHDLAMS.
`nmpVHDLToVHDL87()`	Maps identifier from VHDL to VHDL87.
`nmpVHDLAMSToCDBA()`	Maps identifier from VHDLAMS to CDBA.
`nmpVHDLAMSToConcept()`	Maps identifier from VHDLAMS to Concept.
`nmpVHDLAMSToDef()`	Maps identifier from VHDLAMS to Def.
`nmpVHDLAMSToGcf()`	Maps identifier from VHDLAMS to Gcf.
`nmpVHDLAMSToGenesis()`	Maps identifier from VHDLAMS to Genesis.
`nmpVHDLAMSToLef()`	Maps identifier from VHDLAMS to Lef.
`nmpVHDLAMSToPrint()`	Maps identifier from VHDLAMS to Print.
`nmpVHDLAMSToSdf()`	Maps identifier from VHDLAMS to Sdf.
`nmpVHDLAMSToSpf()`	Maps identifier from VHDLAMS to Spf.
`nmpVHDLAMSToSpectre()`	Maps identifier from VHDLAMS to Spectre.
`nmpVHDLAMSToSpectreHDL()`	Maps identifier from VHDLAMS to SpectreHDL.
`nmpVHDLAMSToSpef()`	Maps identifier from VHDLAMS to Spef.
`nmpVHDLAMSToSpice()`	Maps identifier from VHDLAMS to Spice.
`nmpVHDLAMSToSysVerilog()`	Maps identifier from VHDLAMS to SysVerilog.
`nmpVHDLAMSToVerilog()`	Maps identifier from VHDLAMS to Verilog.
`nmpVHDLAMSToVerilogA()`	Maps identifier from VHDLAMS to VerilogA.
`nmpVHDLAMSToVerilogAMS()`	Maps identifier from VHDLAMS to VerilogAMS.
`nmpVHDLAMSToVHDL()`	Maps identifier from VHDLAMS to VHDL.
`nmpVHDLAMSToVHDL87()`	Maps identifier from VHDLAMS to VHDL87.
`nmpVHDL87ToCDBA()`	Maps identifier from VHDL87 to CDBA.
`nmpVHDL87ToConcept()`	Maps identifier from VHDL87 to Concept.
`nmpVHDL87ToDef()`	Maps identifier from VHDL87 to Def.
`nmpVHDL87ToGcf()`	Maps identifier from VHDL87 to Gcf.
`nmpVHDL87ToGenesis()`	Maps identifier from VHDL87 to Genesis.
`nmpVHDL87ToLef()`	Maps identifier from VHDL87 to Lef.
`nmpVHDL87ToPrint()`	Maps identifier from VHDL87 to Print.
`nmpVHDL87ToSdf()`	Maps identifier from VHDL87 to Sdf.

`nmpVHDL87ToSpf()`	Maps identifier from VHDL87 to Spf.
`nmpVHDL87ToSpectre()`	Maps identifier from VHDL87 to Spectre.
`nmpVHDL87ToSpectreHDL()`	Maps identifier from VHDL87 to SpectreHDL.
`nmpVHDL87ToSpef()`	Maps identifier from VHDL87 to Spef.
`nmpVHDL87ToSpice()`	Maps identifier from VHDL87 to Spice.
`nmpVHDL87ToSysVerilog()`	Maps identifier from VHDL87 to SysVerilog.
`nmpVHDL87ToVerilog()`	Maps identifier from VHDL87 to Verilog.
`nmpVHDL87ToVerilogA()`	Maps identifier from VHDL87 to VerilogA.
`nmpVHDL87ToVerilogAMS()`	Maps identifier from VHDL87 to VerilogAMS.
`nmpVHDL87ToVHDL()`	Maps identifier from VHDL87 to VHDL.
`nmpVHDL87ToVHDLAMS()`	Maps identifier from VHDL87 to VHDLAMS.

nmpPath*<NameSpace>*To*<NameSpace>* Functions

The following table lists all current `nmpPath<NameSpace>To<NameSpace>` functions. For information about these functions, see nmpPath*<NameSpace>To<NameSpace>*.

`nmpPathCDBAToVHDL()`	Converts a simple instance vector in the CDBA namespace to the VHDL namespace.	Examples:
		`nmpPathCDBAToVHDL( "i8<0:3>" ) => "i8(0 to 3)"`
		`nmpPathCDBAToVHDL( "I8<0:3>" ) => "\I8\(0 to 3)"`
		`nmpPathCDBAToVHDL( "instName<4>" ) => "\instName\(4)"`
		`nmpPathCDBAToVHDL( "inst_name<4>" ) => "inst_name(4)"`

`nmpPathCDBAToVerilog()`	Converts a simple instance vector in the CDBA namespace to the Verilog namespace.	Examples:
		`nmpPathCDBAToVerilog( "I8<0:3>" ) => "I8[0:3]"`
		`nmpPathCDBAToVerilog( "instName<4>" ) => "instName[4]"`

3

cdsCopy Functions

The chapter provides syntax and descriptions for the Cadence SKILL functions associated with cdsCopy:

- ccpCopy
- ccpCopyDesign
- ccpCopyExactDesign
- ccpCopyConfig
- ccpDmHasRename
- ccpDmRename
- ccpExpand
- ccpExpandDesign
- ccpExpandExactDesign
- ccpExpandConfig
- ccpRename
- ccpRenameReferenceLib
- ccpRegMonitor
- ccpRegTrigger
- ccpRemoveTrigger

Prefixes Used in Argument Names

Argument names in the SKILL functions described in this section have a one-character prefix that denotes the data type of the argument. The prefix is followed by an underscore character (_). For example, `t_libName` indicates the argument is a string, while `l_src` indicates the argument is a list. The prefix and the underscore are just naming conventions; you do not need to type them when you specify the value of the argument.

The following data type prefixes are used in this section:

`l`	list
`s`	symbol
`g`	general
`G`	gdmSpec object
`q`	gdmSpecList object

Additional Information

You need to use some GDM SKILL functions to use the cdsCopy functions. For information about GDM SKILL functions, see Generic Design Management (GDM) Commands.

For information about the SKILL language, see the following manuals:

- *Cadence SKILL Language User Guide*
- *Cadence SKILL Language Reference*
- *Cadence SKILL Development Help*
- *Cadence SKILL Development Reference*
- *Cadence SKILL++ Object System Reference*
- *Cadence Interprocess Communications SKILL Reference*
- *Cadence User Interface SKILL Reference*

ccpCopy

```
ccpCopy(
     q_src
     q_dest
     [ g_overWrite ]
     [ s_expFlag ]
     [ l_copyViewTypeList ]
     [ l_copyViewNameList ]
     [ t_vNameSimExp ]
     [ t_vNameSpace ]
     [ s_whatToUpdate ]
     [ q_updateLibList ]
     [ g_addProp ]
     [ g_existenceCheck ]
     [ g_reReferenceCustomVias ]
     )
     => t / nil
```

Description

Copies source data to another location.

You can expand the data while copying. If you have already expanded the design with the `ccpExpand`, `ccpExpandDesign`, or `ccpExpandConfig` functions, use `CCP_NO_EXPAND` as the value of the *`s_expFlag`* argument.

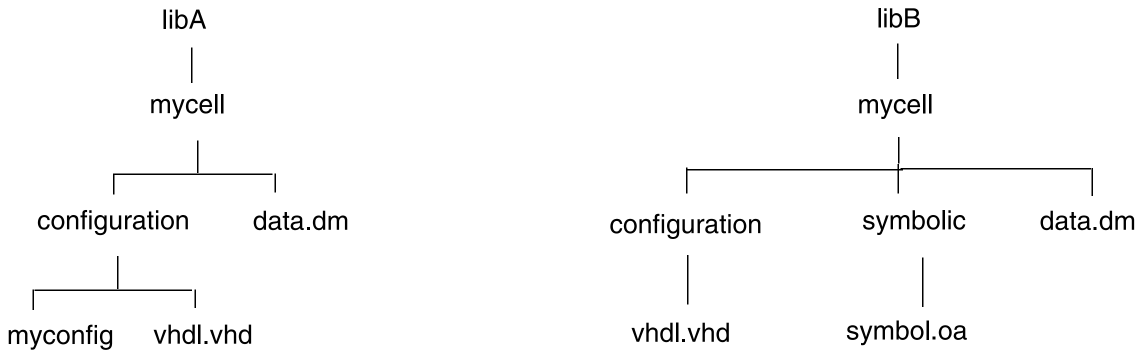
Temporary Directories: If the source data that is being copied is in a library that has a temporary directory assigned, then `ccpCopy` copies files from both the regular library directory and its temporary directory. If the same file exists in both the temporary directory and the library directory, the file in the temporary directory is the one that is copied, regardless of its datestamp or whether it is a source file or a derived file. Files are copied to the temporary directory of the destination library only if there are pre-existing files of the same name in the directory; otherwise they are copied into the regular destination library.

Note: If two objects in the source list have the same name, `ccpCopy` will not copy them to the same location in one copy operation, unless you choose to automatically rename them. See Automatically Renaming Files for more information.

Automatically Renaming Files

If two objects in the source list have the same name, you cannot copy them to the same location in one copy operation, unless you choose to automatically rename them.

For example, if you have the following data:



and you try to copy `libA/mycell` and `libB/mycell` to the same location, nothing will be copied because `libA/mycell/configuration/vhdl.vhd` conflicts with `libB/mycell/configuration/vhdl.vhd` and `libA/mycell/data.dm` conflicts with `libB/mycell/data.dm`. In such a case, you can copy the data only if you rename the objects with conflicting names.

Therefore, it is recommended to perform only a single transaction instead of performing multiple transactions simultaneously. For example, if you are trying to copy `libA/mycell/configuration/vhdl.vhd` with `libB/mycell/configuration/vhdl.vhd` and `libA/mycell/data.dm` then there would be a conflict between these two transactions.

To automatically rename objects with conflicting names,

➤ Set the following environment variable:

```
CDS_COPY_AUTO_RENAME "yes"
```

If you set this variable, cells containing the view files that have a name conflict are automatically renamed in the destination location, as mentioned below.

Note: Only the views that have at least one file with a name conflict will trigger the auto renaming of the cell. The other views of the cell, when copied in the same operation, will also be copied to the same renamed cell.The auto renamed cell name may be generated by appending the name of the From Cell (if different), and the name of the From Library (if different). If the cell by the same name already exists, then a unique number is appended to the name. The name additions are separated by an underscore ("_") character.

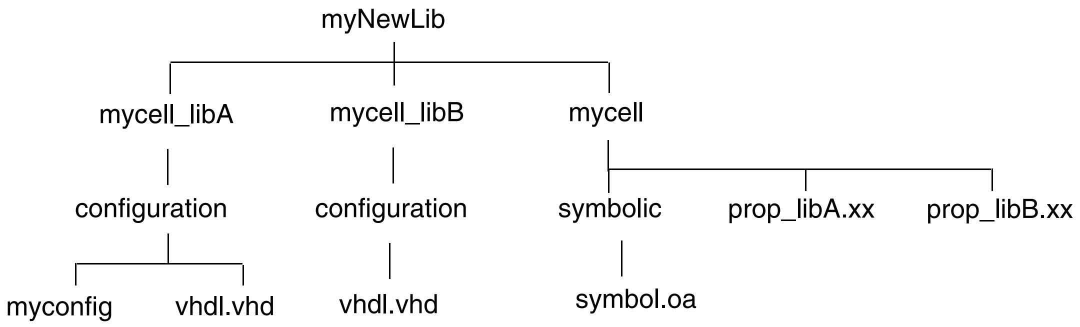
Important

The `CDS_COPY_AUTO_RENAME` environment variable must be set before starting the "Copy" process because the setting is evaluated only during initialization.

Similarly, other library-level files (files in a library that are not cells) and cell-level files (files in a cell that are not views) are also renamed if they have a name conflict. These files are renamed in the following way:

```
file -> file_libName
file.xx -> file_libName.xx
.file -> .file_libName
```

In the above example, if you set `CDS_COPY_AUTO_RENAME` to `"yes"` and then copy `libA/mycell` and `libB/mycell` to `myNewLib`, you get the following files in the destination:



The `CDS_COPY_AUTO_RENAME` variable applies to the `ccpCopy`, `ccpCopyDesign`, and `ccpCopyConfig` functions. However, it only applies when expansion is being done with these functions; it does not apply if *`s_expFlag`* is `CCP_NO_EXPAND`.

You can set the maximum cell name length allowed by using the `CDS_MAX_CELL_NAME_LENGTH` environment variable, which applies to the `ccpCopy`, `ccpCopyDesign`, and `ccpCopyConfig` functions.

Note: The compression level of any copied library data gets adjusted to match the compression level of the destination library.

Arguments

q_src	Source gdmSpecList object containing the gdmSpec objects you want to copy. Each gdmSpec object represents a library, cell, view, or file. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
q_dest	Destination gdmSpecList object containing gdmSpec objects, each of which represents a library, cell, or view. The destination list must have the same number of gdmSpec objects as the source list and each gdmSpec object must be of the same type as its corresponding object in the source list (for example, both must be libraries or both must be cells). Otherwise, the destination list must contain only one object (typically a library, or a cell when the source list contains views). For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
g_overWrite	Boolean overwrite flag. Its value can be `t` or `nil`. (Any other value is considered `t`.) Overwrites the object if it already exists at the destination. If *g_overWrite* is `t` and the object is managed and not already checked out, it is checked out by the system. This argument is optional; its default value is `nil`.

s_expFlag

Expand option. Can be one of the following predefined symbols:

- `CCP_NO_EXPAND:` No expansion is done. Typically used after a call to `ccpExpand`.
- `CCP_EXPAND_COMANAGED`: Expands a directory and includes only the `coManaged` and `alsoManaged` files, which are defined in the data registry.

Note: Some applications have co-managed files stored at different directory levels. These co-managed files will also be included for copy.

Note: The `data.dm` file, which is invalid in OpenAccess libraries, is no longer automatically included in co-managed sets.

- `CCP_EXPAND_ALL`: Expands everything in a directory.

This argument is optional; its default value is `CCP_NO_EXPAND`.

l_copyViewTypeList

List of strings specifying the view types to copy. If the list is empty, includes all view types in the hierarchy. This option is ignored if you use `CCP_NO_EXPAND` as the value of the *s_expFlag* argument.

This argument is optional; its default value is `nil`.

l_copyViewNameList

List of view names to copy. If the list is empty, includes all view names in the hierarchy. This option is ignored if you use `CCP_NO_EXPAND` as the value of the *s_expFlag* argument.

This argument is optional; its default value is `nil`.

t_vNameSimExp

Regular expression specifying the views to copy. This option is ignored if you use `CCP_NO_EXPAND` as the value of the *s_expFlag* argument.

Note: This is not a regular expression.

Note: `CCP_NO_EXPAND` is also always invalid for the view-based design and config copies.

`t_vNameSpace`	The name space in which the arguments *`t_vNameSimExp`* and *`l_copyViewNameList`* are provided. Can be one of the following strings: `VHDL`, `Verilog`, `Verilog-A`, `CDBA`, `Concept`, `Library`, `LibraryUnix`, or `LibraryNT`.. For more information about Cadence name spaces, see Name Spaces for Different Data Types.
`s_whatToUpdate`	One of the following predefined symbols, which tells the cross-reference updater what to update: ■ `CCP_UPDATE_COPIED_DATA`: Updates cross-references in only the copied data. ■ `CCP_UPDATE_DESTLIB_ONLY`: Updates all cross-references in the destination library. ■ `CCP_UPDATE_FROM_LIBLIST`: Updates cross-references in the libraries you specify in the next argument, *`q_updateLibList`*. This argument is optional; its default value is `CCP_UPDATE_COPIED_DATA`. **Note:** There might be a scenario where the entire destination library containing a cellview is selected for copy updates. In addition, the cellviews already existing in the destination library will also have their data references. In that case, the updates need to be made in more files other than just the copied set.
`q_updateLibList`	Valid only if the *`s_whatToUpdate`* argument has a value of `CCP_UPDATE_FROM_LIBLIST`. Otherwise, this argument is ignored. *`q_updateLibList`* takes a gdmSpecList object containing library gdmSpec objects, which specify the libraries in which to update cross-references. **Note:** If *`q_updateLibList`* is empty and *`s_whatToUpdate`* is `CCP_UPDATE_FROM_LIBLIST`, copy proceeds but no updating is done. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.

g_addProp	`t` or `nil`; default is `nil`. If you specify `t`, library and cell property files are also copied. It is recommended that you use `nil` for this argument because of potential conflicts between property files. Properties are usually added by the copy updaters.
g_existenceCheck	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to validate the existence of layers, purposes, viaDefs, and siteDefs in the technology database of the destination library after the copy command is completed.
g_reReferenceCustomVias	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to update custom via definitions to point to cellviews in the destination library.

Value Returned

`t`	The source data was copied.
`nil`	The source data was not copied.

Example

See "cdsCopy SKILL Examples" on page 142.

ccpCopyDesign

```
ccpCopyDesign(
     G_src
     G_dest
     [ g_overWrite ]
     [ s_expFlag ]
     [ q_copySkipLibList ]
     [ l_copyViewTypeList ]
     [ l_copyViewNameList ]
     [ t_vNameSimExp ]
     [ t_vNameSpace ]
     [ s_whatToUpdate ]
     [ q_updateLibList ]
     [ g_addProp ]
     [ g_existenceCheck ]
     [ g_reReferenceCustomVias ]
     )
     => t / nil
```

Description

Copies a design, after expanding it, to the destination library you specify.

`ccpCopyDesign` first traverses the top level design, which is specified by the source cell or view *`G_src`*. It uses the pcdb (parent/child database) API to find all references to other cells. It then traverses those cells, except the ones that are in libraries listed in *`q_copySkipLibList`*. `ccpCopyDesign` then copies the data, using the following filtering process:

- It copies only the views listed in *`l_copyViewTypeList`*, *`l_CopyViewNameList`*, and *`t_vNameSimExp`*. If the lists are empty, it copies all views.
- It copies files based upon the value of the *`s_expFlag`* argument, which determines whether all files or only the `coManaged` and `alsoManaged` files are copied.

Temporary Directories: If the source data that is being copied is in a library that has a temporary directory assigned, then `ccpCopyDesign` copies files from both the regular library directory and its temporary directory. If the same file exists in both the temporary directory and the library directory, the file in the temporary directory is the one that is copied, regardless of its datestamp or whether it is a source file or a derived file. Files are copied to the temporary directory of the destination library only if there are pre-existing files of the same name in the directory; otherwise they are copied into the regular destination library.

Name Conflicts: If two objects in the source list have the same name, `ccpCopyDesign` will not copy them to the same location in one copy operation, unless you choose to automatically rename them. See the description of ccpCopy for more information.

Arguments

`G_src`	Source gdmSpec object. Must be a cell or view. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
`G_dest`	Destination gdmSpec object. Must be a library. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
`g_overWrite`	Boolean overwrite flag. Its value can be `t` or `nil`. (Any other value is considered `t`.) Overwrites the object if it already exists at the destination. If *`g_overWrite`* is `t` and the object is managed and not already checked out, it is checked out by the system.
`s_expFlag`	Expand option. Can be one of the following predefined symbols: ■ `CCP_EXPAND_COMANAGED`: Expands a directory and includes only the `coManaged` and `alsoManaged` files, which are defined in the `dataReg`. **Note:** Some applications have co-managed files stored at different directory levels. These co-managed files will also be included for copy. **Note:** The `prop.xx` file, which is invalid in OpenAccess libraries, is no longer automatically included in co-managed sets. ■ `CCP_EXPAND_ALL`: Expands everything in a directory.
`q_copySkipLibList`	gdmSpecList object containing the libraries to exclude while copying. *`q_copySkipLibList`* is a stop list—design objects in the libraries are not traversed. For information about creating gdmSpec and gdmSpecList objects, Generic Design Management (GDM) Functions. This argument is optional; its default value is `nil`. *`l_copyViewTypeList`* List of strings specifying the view types to copy. If the list is empty, includes all view types in the hierarchy. This argument is optional; its default value is `nil`.

`l_copyViewNameList`

List of view names to copy. If the list is empty, includes all view names in the hierarchy.

This argument is optional; its default value is `nil`.

`t_vNameSimExp` Regular expression specifying the views to copy. This option is ignored if you use `CCP_NO_EXPAND` as the value of the *`s_expFlag`* argument.

`t_vNameSpace` The name space in which the arguments *`t_vNameSimExp`* and *`l_copyViewNameList`* are provided. Can be one of the following strings: `VHDL`, `Verilog`, `Verilog-A`, `CDBA`, `Concept`, `Library`, `LibraryUnix`, or `LibraryNT`.

For more information about Cadence name spaces, see

Name Spaces for Different Data Types.

`s_whatToUpdate` One of the following predefined symbols, which tells the cross-reference updater what to update:

- `CCP_UPDATE_COPIED_DATA`: Updates cross-references in only the copied data.
- `CCP_UPDATE_DESTLIB_ONLY`: Updates all cross-references in the destination library.
- `CCP_UPDATE_FROM_LIBLIST`: Updates cross-references in the libraries you specify in the *`q_updateLibList`* argument.

This argument is optional; its default value is `CCP_UPDATE_COPIED_DATA`.

q_updateLibList	Valid only if the *s_whatToUpdate* argument has a value of `CCP_UPDATE_FROM_LIBLIST`. Otherwise, this argument is ignored. *q_updateLibList* takes a gdmSpecList object containing library gdmSpec objects, which specify the libraries in which to update cross-references. **Note:** If *q_updateLibList* is empty and *s_whatToUpdate* is `CCP_UPDATE_FROM_LIBLIST`, copy proceeds but no updating is done. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
g_addProp	`t` or `nil`; default is `nil`. If you specify `t`, library and cell property files are also copied. It is recommended that you use `nil` for this argument because of potential conflicts between property files. Properties are usually added by the copy updaters.
g_existenceCheck	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to validate the existence of layers, purposes, viaDefs, and siteDefs in the technology database of the destination library after the copy command is completed.
g_reReferenceCustomVias	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to update custom via definitions to point to cellviews in the destination library.

Value Returned

`t`	The design was copied.
`nil`	The design was not copied.

ccpCopyExactDesign

```
ccpCopyExactDesign(
     G_src
     G_dest
     [ g_overWrite ]
     [ s_expFlag ]
     [ q_copySkipLibList ]
     [ l_copyViewTypeList ]
     [ l_copyViewNameList ]
     [ t_vNameSimExp ]
     [ t_vNameSpace ]
     [ s_whatToUpdate ]
     [ q_updateLibList ]
     [ g_addProp ]
     [ g_existenceCheck ]
     [ g_reReferenceCustomVias ]
     )
     => t / nil
```

Description

Copies a design, after expanding it, to the destination library you specify. This function is similar to `ccpCopyDesign`, except that it copies only the exact design—it does not copy any views that are not used in the design.

`ccpCopyExactDesign` traverses the design hierarchy based on the pcdb (parent-child database). It excludes the cells that are in libraries listed in *`q_copySkipLibList`*, if any. It then copies the data.

Note: For each cell, `ccpCopyExactDesign` copies only the view that is used in the design. It does not copy any other views.

To copy other views, you need to specify them in the *`l_copyViewTypeList`* or *`l_copyViewNameList`*. If these views have their own hierarchy, then that hierarchy is also traversed.

Temporary Directories: If the source data that is being copied is in a library that has a temporary directory assigned, then `ccpCopyExactDesign` copies files from both the regular library directory and its temporary directory. If the same file exists in both the temporary directory and the library directory, the file in the temporary directory is the one that is copied, regardless of its datestamp or whether it is a source file or a derived file. Files are copied to the temporary directory of the destination library only if there are pre-existing files of the same name in the directory; otherwise they are copied into the regular destination library.

Name Conflicts: If two objects in the source list have the same name, `ccpCopyExactDesign` will not copy them to the same location in one copy operation, unless you choose to automatically rename them. See the description of ccpCopy for more information.

Arguments

`G_src`	Source gdmSpec object. Must be a cell or cellview. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
`G_dest`	Destination gdmSpec object. Must be a library. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
`g_overWrite`	Boolean overwrite flag. Its value can be `t` or `nil`. (Any other value is considered `t`.) Overwrites the object if it already exists at the destination. If *`g_overWrite`* is `t` and the object is managed and not already checked out, it is checked out by the system.
`s_expFlag`	Expand option. Can be one of the following predefined symbols: ■ `CCP_EXPAND_COMANAGED`: Expands a directory and includes only the `coManaged` and `alsoManaged` files, which are defined in the `dataReg`. **Note:** Some applications have co-managed files stored at different directory levels. These co-managed files will also be included for copy. **Note:** The `prop.xx` file, which is invalid in OpenAccess libraries, is no longer automatically included in co-managed sets. ■ `CCP_EXPAND_ALL`: Expands everything in a directory.
`q_copySkipLibList`	gdmSpecList object containing the libraries to exclude while copying. *`q_copySkipLibList`* is a stop list—design objects in the libraries are not traversed. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions. This argument is optional; its default value is `nil`. *`l_copyViewTypeList`* List of strings specifying additional view types to copy. If the list is empty, only the views that are used in the design are copied. This argument is optional; its default value is `nil`.

`l_copyViewNameList`	
	List of strings specifying additional view names to copy. If the list is empty, only the views that are used in the design are copied. This argument is optional; its default value is `nil`.
`t_vNameSimExp`	Regular expression specifying additional views to copy. This option is ignored if you use `CCP_NO_EXPAND` as the value of the *`s_expFlag`* argument.
`t_vNameSpace`	The name space in which the arguments *`t_vNameSimExp`* and *`l_copyViewNameList`* are provided. For more information about Cadence name spaces, see Name Spaces for Different Data Types.
`s_whatToUpdate`	One of the following predefined symbols, which tells the cross-reference updater what to update: ■ `CCP_UPDATE_COPIED_DATA`: Updates cross-references in only the copied data. ■ `CCP_UPDATE_DESTLIB_ONLY`: Updates all cross-references in the destination library. ■ `CCP_UPDATE_FROM_LIBLIST`: Updates cross-references in the libraries you specify in the *`q_updateLibList`* argument. This argument is optional; its default value is `CCP_UPDATE_COPIED_DATA`.
`q_updateLibList`	Valid only if the *`s_whatToUpdate`* argument has a value of `CCP_UPDATE_FROM_LIBLIST`. Otherwise, this argument is ignored. *`q_updateLibList`* takes a gdmSpecList object containing library gdmSpec objects, which specify the libraries in which to update cross-references. **Note:** If *`q_updateLibList`* is empty and *`s_whatToUpdate`* is `CCP_UPDATE_FROM_LIBLIST`, copy proceeds but no updating is done. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.

g_addProp	`t` or `nil`; default is `nil`. If you specify `t`, library and cell property files are also copied. It is recommended that you use `nil` for this argument because of potential conflicts between property files. Properties are usually added by the copy updaters.
g_existenceCheck	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to validate the existence of layers, purposes, viaDefs, and siteDefs in the technology database of the destination library after the copy command is completed.
g_reReferenceCustomVias	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to update custom via definitions to point to cellviews in the destination library.

Value Returned

`t`	The design was copied.
`nil`	The design was not copied.

ccpCopyConfig

```
ccpCopyConfig(
     G_src
     G_dest
     [ g_overWrite ]
     [ s_expFlag ]
     [ q_copySkipLibList ]
     [ l_copyViewTypeList ]
     [ l_copyViewNameList ]
     [ t_vNameSimExp ]
     [ t_vNameSpace ]
     [ s_whatToUpdate ]
     [ q_updateLibList ]
     [ g_addProp ]
     [ g_existenceCheck ]
     [ g_reReferenceCustomVias ]
     )
     => t / nil
```

Description

Copies a design configuration. Expansion is based on configuration rules specified in the `expand.cfg` file in the source configuration view.

A configuration is a set of rules that defines which cellviews under a top-level cell are to be considered part of the design for a given purpose such as netlisting or simulation. For more information about configurations, see the *Cadence Hierarchy Editor User Guide*.

Temporary Directories: If the source data that is being copied is in a library that has a temporary directory assigned, then `ccpCopyConfig` copies files from both the regular library directory and its temporary directory. If the same file exists in both the temporary directory and the library directory, the file in the temporary directory is the one that is copied, regardless of its datestamp or whether it is a source file or a derived file. Files are copied to the temporary directory of the destination library only if there are pre-existing files of the same name in the directory; otherwise they are copied into the regular destination library.

Name Conflicts: If two objects in the source list have the same name, `ccpCopyConfig` will not copy them to the same location in one copy operation, unless you choose to automatically rename them. See the description of ccpCopy for more information.

Arguments

G_src	Source gdmSpec object, which represents a cell or a configuration view. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
G_dest	Destination gdmSpec object. Must be a library. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
g_overWrite	Boolean overwrite flag. Its value can be `t` or `nil`. (Any other value is considered `t`.) Overwrites the object if it already exists at the destination. If *g_overWrite* is `t` and the object is managed and not already checked out, it is checked out by the system.
s_expFlag	Expand option. Can be one of the following predefined symbols: ■ `CCP_EXPAND_COMANAGED`: Expands a directory and includes only the `coManaged` and `alsoManaged` files, which are defined in the `dataReg`. **Note:** Some applications have co-managed files stored at different directory levels. These co-managed files will also be included for copy. **Note:** The `prop.xx` file, which is invalid in OpenAccess libraries, is no longer automatically included in co-managed sets. ■ `CCP_EXPAND_ALL`: Expands everything in a directory.

q_copySkipLibList gdmSpecList object containing the libraries to exclude while copying. *q_copySkipLibList* is a stop list—design objects in the libraries are not traversed.

`For information about creating gdmSpec and gdmSpecList objects, see` Generic Design Management (GDM) Functions.

This argument is optional; its default value is `nil`.

l_copyViewTypeList List of strings specifying the view types to copy. If the list is empty, includes all view types in the hierarchy.

This argument is optional; its default value is `nil`.

l_copyViewNameList

List of view names to copy. If the list is empty, includes all view names in the hierarchy.

This argument is optional; its default value is `nil`.

t_vNameSimExp Regular expression specifying the views to copy. This option is ignored if you use `CCP_NO_EXPAND` as the value of the *s_expFlag* argument.

t_vNameSpace The name space in which the arguments *t_vNameSimExp* and *l_copyViewNameList* are provided. Can be one of the following strings: `VHDL`, `Verilog`, `Verilog-A`, `CDBA`, `Concept`, `Library`, `LibraryUnix`, or `LibraryNT`.

For more information about Cadence name spaces, see Name Spaces for Different Data Types.

s_whatToUpdate	One of the following predefined symbols, which tells the cross-reference updater what to update: ■ `CCP_UPDATE_COPIED_DATA`: Updates cross-references in only the copied data. ■ `CCP_UPDATE_DESTLIB_ONLY`: Updates all cross-references in the destination library. ■ `CCP_UPDATE_FROM_LIBLIST`: Updates cross-references in the libraries you specify in the *_updateLibList* argument. This argument is optional; its default value is `CCP_UPDATE_COPIED_DATA`.
q_updateLibList	Valid only if the *s_whatToUpdate* argument has a value of `CCP_UPDATE_FROM_LIBLIST`. Otherwise, this argument is ignored. *q_updateLibList* takes a gdmSpecList object containing library gdmSpec objects, which specify the libraries in which to update cross-references. **Note:** If *q_updateLibList* is empty and *s_whatToUpdate* is `CCP_UPDATE_FROM_LIBLIST`, copy proceeds but no updating is done. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
g_addProp	`t` or `nil`; default is `nil`. If you specify `t`, library and cell property files are also copied. It is recommended that you use `nil` for this argument because of potential conflicts between property files. Properties are usually added by the copy updaters.
g_existenceCheck	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to validate the existence of layers, purposes, viaDefs, and siteDefs in the technology database of the destination library after the copy command is completed.
g_reReferenceCustomVias	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to update custom via definitions to point to cellviews in the destination library.

Value Returned

`t`	The design configuration was copied.
`nil`	The design configuration was not copied.

ccpDmHasRename

```
ccpDmHasRename(
     t_dmName
     )
     => t / nil
```

Description

Checks whether a specified design management system (DMS) is loaded and enabled for GDM renaming. The DMS can be loaded either by Virtuoso at startup or by previously made successful calls to the DM objects from SKILL functions such as `ddGetObjDMSys`, `ddIsPathManaged`, or `gdmstatus`.

Arguments

`t_dmName`	The GDM name of the DMS as returned by the `ddGetObjDMSys` function. See also Generic Design Management (GDM) Functions.

Value Returned

`t`	The DMS specified was found to be successfully loaded and enabled for GDM renaming.
`nil`	The DMS specified was not found to be loaded or is not enabled for GDM renaming.

ccpDmRename

```
ccpDmRename(
     G_src
     G_dest
     [ ?destLibPath t_destLibPath ]
     [ ?whatToUpdate s_whatToUpdate ]
     [ ?updateLibList q_updateLibList ]
     [ ?existenceCheck x_existenceCheck ]
     [ ?reReferenceCustomVias x_reReferenceCustomVias ]
     [ ?tag t_tag ]
     [ ?description t_description ]
     [ ?xtraArgs t_xtraArgs ]
     [ ?feedbackStyle x_feedbackStyle ]
     [ ?limitChecks x_limitChecks ]
     [ ?checkOnly g_checkOnly ]
     [ ?returnGdmContext g_returnGdmContext ]
     [ ?returnCcpResults g_returnCcpResults ]
     [ ?feedbackCallback u_feedbackCallback ]
     [ ?doneCallback u_doneCallback ]
     [ ?gdmContextCallback u_gdmContextCallback ]
     [ ?nconsTag l_nconsTag ]
     [ ?dpl l_dpl ]
     )
     => t / nil
```

Description

Renames a specified gdmSpec object representing a cellview. The source object is deleted after a copy is created. When you copy a directory containing the cellview, all files in the directory are copied, regardless of whether they are co-managed or not.

Note: You can set the maximum cell name length allowed by using the `CDS_MAX_CELL_NAME_LENGTH` environment variable.

Arguments

`G_src`	Source gdmSpec object representing a cellview. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
`G_dest`	Destination gdmSpec object. Must be of the same type as the source object.

`?destLibPath` *`t_destLibPath`*

When renaming creates a new library, optionally specify the path for the library. The parent directory specified in the path must already exist.

`?whatToUpdate` *`s_whatToUpdate`*

One of the following predefined symbols that tell the cross-reference updater what to update:

- `CCP_UPDATE_COPIED_DATA`: Updates cross-references in only the renamed data.
- `CCP_UPDATE_DESTLIB_ONLY`: Updates all cross-references in the destination library. This is the default value.
- `CCP_UPDATE_FROM_LIBLIST`: Updates cross-references in the libraries you specify in the next argument, *`q_updateLibList`*.

`?updateLibList` *`q_updateLibList`*

Valid only if the *`s_whatToUpdate`* argument has a value of `CCP_UPDATE_FROM_LIBLIST`. Otherwise, this argument is ignored.

`q_updateLibList` takes a gdmSpecList object containing library gdmSpec objects, which specify the libraries in which to update cross-references.

Note: If *`q_updateLibList`* is empty and *`s_whatToUpdate`* is `CCP_UPDATE_FROM_LIBLIST`, rename proceeds but no updating is done.

For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.

`?existenceCheck` *`x_existenceCheck`*

`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to validate the existence of layers, purposes, viaDefs, and siteDefs in the technology database of the destination library after the copy command is completed.

`?reReferenceCustomVias` *`x_reReferenceCustomVias`*

`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to update custom via definitions to point to cellviews in the destination library.

`?tag` *`t_tag`* String to return callback functions and final return values.

`?description` *`t_description`*

String for the DM comment provided while renaming.

`?xtraArgs` *`t_xtraArgs`*

String for custom optional arguments of the DMS.

`?feedbackStyle` *`x_feedbackStyle`*

Set of bit flags.

`?limitChecks` *`x_limitChecks`*

Set of bit flags.

`?checkOnly` *`g_checkOnly`*

Checking mode. When set to `t`, it checks for permissions without renaming. When set to `nil`, it checks for permissions and then renames when no error is found.

`?returnGdmContext` *`g_returnGdmContext`*

GDM rename context from calling functions.

`?returnCcpResults` *`g_returnCcpResults`*

Final return value details of the rename, as a disembodied property list (DPL)

`?feedbackCallback` *`u_feedbackCallback`*

Callbacks indicating that specific DM rename phases have run.

`?doneCallback` *`u_doneCallback`*

The one-time callback when rename is done, before returning to the calling function.

`?gdmContextCallback` *`u_gdmContextCallback`*

	The callback after the GDM context has been defined for the rename operation.
`?nconsTag` *`l_nconsTag`*	
	The object tag to return with callback functions.
`?dpl` *`l_dpl`*	A disembodied property list (DPL) of arguments that can be assigned, overriding keyword arguments.

Value Returned

`t`	The object was renamed, and related tasks, such as preserving DM history, were completed successfully.
`nil`	The object could not be renamed, or one or more of the related tasks failed.

ccpExpand

```
ccpExpand(
     q_src
     s_expFlag
     [ l_expandViewTypeList ]
     [ l_expandViewNameList ]
     [ t_vNameSimExp ]
     [ t_vNameSpace ]
     [ g_addProp ]
     [ g_existenceCheck ]
     [ g_reReferenceCustomVias ]
     )
     => q_expSpecList / nil
```

Description

Does an expansion, given a list of source objects.

Arguments

q_src	gdmSpecList object containing the gdmSpec objects you want to expand. Each gdmSpec object represents a library, cell, or view. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
s_expFlag	Expand option. Can be one of the following predefined symbols: ■ `CCP_EXPAND_COMANAGED`: Expands a directory and includes only the `coManaged` and `alsoManaged` files, which are defined in the `dataReg`. **Note:** Some applications have co-managed files stored at different directory levels. These co-managed files will also be included for copy. **Note:** The `prop.xx` file, which is invalid in OpenAccess libraries, is no longer automatically included in co-managed sets. ■ `CCP_EXPAND_ALL`: Expands everything in a directory.
l_expandViewTypeList	List of strings specifying the view types to be expanded. If the list is empty, includes all view types in the hierarchy. This argument is optional; its default value is `nil`.
l_expandViewNameList	List of view names to be expanded. If the list is empty, includes all view names in the hierarchy. This argument is optional; its default value is `nil`.
t_vNameSimExp	Regular expression specifying the views to expand.
t_vNameSpace	The name space in which the arguments *t_vNameSimExp* and *l_expandViewNameList* are provided. Can be one of the following strings: `VHDL`, `Verilog`, `Verilog-A`, `CDBA`, `Concept`, `Library`, `LibraryUnix`, or `LibraryNT`.. For more information about Cadence name spaces, see Name Spaces for Different Data Types.

g_addProp	`t` or `nil`; default is `nil`. If you specify `t`, library and cell property files are also copied. It is recommended that you use `nil` for this argument because of potential conflicts between property files. Properties are usually added by the copy updaters.
g_existenceCheck	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to validate the existence of layers, purposes, viaDefs, and siteDefs in the technology database of the destination library after the copy command is completed. **Note:** This argument is typically used with the copy functions and is not needed for expansion functions.
g_reReferenceCustomVias	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to update custom via definitions to point to cellviews in the destination library. **Note:** This argument is typically used with the copy functions and is not needed for expansion functions.

Value Returned

q_expSpecList	A gdmSpecList object containing gdmSpec objects for the files in the expansion list.
`nil`	The source objects could not be expanded.

ccpExpandDesign

```
ccpExpandDesign(
     G_src
     s_expFlag
     q_expandSkipLibList
     [ l_expandViewTypeList ]
     [ l_expandViewNameList ]
     [ t_vNameSimExp ]
     [ t_vNameSpace ]
     [ g_addProp ]
     [ g_existenceCheck ]
     [ g_reReferenceCustomVias ]
     )
     => q_expSpecList / nil
```

Description

Expands a design, given a source cell or cellview gdmSpec object.

`ccpExpandDesign` first traverses the top level design, which is specified by the source cell or view *`G_src`*. It uses the pcdb (parent/child database) API to find all references to other cells. It then traverses those cells, except the ones that are in libraries listed in *`q_expandSkipLibList`*. `ccpExpandDesign` then creates the expansion list, using the following filtering process:

- It includes only the views listed in *`l_expandViewTypeList`*, *`l_expandViewNameList`*, and *`t_vNameSimExp`*. If the lists are empty, it includes all views.
- It includes files based upon the value of the *`s_expFlag`* argument, which determines whether all files or only the `coManaged` and `alsoManaged` files are included.

Arguments

G_src	Source gdmSpec object. Must be a cell or view. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
s_expFlag	Expand option. Can be one of the following predefined symbols: ■ `CCP_EXPAND_COMANAGED`: Expands a directory and includes only the `coManaged` and `alsoManaged` files, which are defined in the `dataReg`. **Note:** Some applications have co-managed files stored at different directory levels. These co-managed files will also be included for copy. **Note:** The `prop.xx` file, which is invalid in OpenAccess libraries, is no longer automatically included in co-managed sets. `CCP_EXPAND_ALL`: Expands everything in a directory.
q_expandSkipLibList	gdmSpecList object is provided that contains the libraries, which are excluded while copying. *q_expandSkipLibList* is a stop list—design objects in the libraries are not traversed. For more information about creating gdmSpec and gdmSpecList objects, see gdmCreateSpec and gdmCreateSpecList.
l_expandViewTypeList	List of strings specifying the view types to be expanded. If the list is empty, includes all view types in the hierarchy. This argument is optional; its default value is `nil`.
l_expandViewNameList	List of view names to be expanded. If the list is empty, includes all view names in the hierarchy. This argument is optional; its default value is `nil`.
t_vNameSimExp	Regular expression specifying the views to expand.

t_vNameSpace	The name space in which the arguments *t_vNameSimExp* and *l_expandViewNameList* are provided. Can be one of the following strings: `VHDL`, `Verilog`, `Verilog-A`, `CDBA`, `Concept`, `Library`, `LibraryUnix`, or `LibraryNT`.. For a list of Cadence name spaces, see Name Spaces for Different Data Types.
g_addProp	`t` or `nil`; default is `nil`. If you specify `t`, library and cell property files are also copied. It is recommended that you use `nil` for this argument because of potential conflicts between property files. Properties are usually added by the copy updaters.
g_existenceCheck	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to validate the existence of layers, purposes, viaDefs, and siteDefs in the technology database of the destination library after the copy command is completed. **Note:** This argument is typically used with the copy functions and is not needed for expansion functions.
g_reReferenceCustomVias	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to update custom via definitions to point to cellviews in the destination library. **Note:** This argument is typically used with the copy functions and is not needed for expansion functions.

Value Returned

q_expSpecList	A gdmSpecList object containing gdmSpec objects for the files in the expansion list.
`nil`	The design could not be expanded.

ccpExpandExactDesign

```
ccpExpandExactDesign(
     G_src
     s_expFlag
     q_expandSkipLibList
     [ l_expandViewTypeList ]
     [ l_expandViewNameList ]
     [ t_vNameSimExp ]
     [ t_vNameSpace ]
     [ g_addProp ]
     [ g_existenceCheck ]
     [ g_reReferenceCustomVias ]
     )
     => q_expSpecList / nil
```

Description

Expands a design, given a source cell or cellview gdmSpec object. This function is similar to `ccpExpandDesign`, except that the expansion list only includes those views that are used in the design.

`ccpExpandExactDesign` traverses the design hierarchy based on the pcdb (parent-child database). It excludes the cells that are in libraries listed in *`q_expandSkipLibList`*, if any. It then returns the expansion list.

Note: For each cell, `ccpExpandExactDesign` includes only the view that is used in the design. It does not include any other views in the expansion list.

To include other views, you need to specify them in the *`l_expandViewTypeList`* or *`l_expandViewNameList`*. If these views have their own hierarchy, then that hierarchy is also traversed.

Arguments

G_src	Source gdmSpec object. Must be a cell or view. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
s_expFlag	Expand option. Can be one of the following predefined symbols: ■ `CCP_EXPAND_COMANAGED`: Expands a directory and includes only the `coManaged` and `alsoManaged` files, which are defined in the `dataReg`. **Note:** Some applications have co-managed files stored at different directory levels. These co-managed files will also be included for copy. **Note:** The `prop.xx` file, which is invalid in OpenAccess libraries, is no longer automatically included in co-managed sets. ■ `CCP_EXPAND_ALL`: Expands everything in a directory.
q_expandSkipLibList	gdmSpecList object is provided that contains the libraries, which are excluded while copying. *q_expandSkipLibList* is a stop list—design objects in the libraries are not traversed. For more information about creating gdmSpec and gdmSpecList objects, see gdmCreateSpec and gdmCreateSpecList.
l_expandViewTypeList	List of strings specifying the additional view types to be expanded. If the list is empty, includes only those views that are used in the design. This argument is optional; its default value is `nil`.
l_expandViewNameList	List of strings specifying the additional views to be expanded. If the list is empty, includes only those views that are used in the design. This argument is optional; its default value is `nil`.

t_vNameSimExp	Regular expression specifying the additional views to expand.
t_vNameSpace	The name space in which the arguments *t_vNameSimExp* and *l_expandViewNameList* are provided. Can be one of the following strings: `VHDL`, `Verilog`, `Verilog-A`, `CDBA`, `Concept`, `Library`, `LibraryUnix`, or `LibraryNT`.. For a list of Cadence name spaces, see Name Spaces for Different Data Types.
g_addProp	`t` or `nil`; default is `nil`. If you specify `t`, library and cell property files are also copied. It is recommended that you use `nil` for this argument because of potential conflicts between property files. Properties are usually added by the copy updaters.
g_existenceCheck	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to validate the existence of layers, purposes, viaDefs, and siteDefs in the technology database of the destination library after the copy command is completed.
g_reReferenceCustomVias	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to update custom via definitions to point to cellviews in the destination library.

Value Returned

g_expSpecList	A gdmSpecList object containing gdmSpec objects for the files in the expansion list.
`nil`	The design could not be expanded.

ccpExpandConfig

```
ccpExpandConfig(
     G_src
     s_expFlag
     [ q_expandSkipLibList ]
     [ l_expandViewTypeList ]
     [ l_expandViewNameList ]
     [ t_vNameSimExp ]
     [ t_vNameSpace ]
     [ g_addProp ]
     [ g_existenceCheck ]
     [ g_reReferenceCustomVias ]
     )
     => q_expSpecList / nil
```

Description

Expands a configuration, based on configuration rules specified in the `expand.cfg` file in the source configuration view.

A configuration is a set of rules that defines which cellviews under a top-level cell are to be considered part of the design for a given purpose such as netlisting or simulation. For more information about configurations, see the *Hierarchy Editor User Guide*.

Arguments

G_src

Source gdmSpec object which represents a cell or a configuration view.

For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.

s_expFlag

Expand option. Can be one of the following predefined symbols:

- `CCP_EXPAND_COMANAGED`: Expands a directory and includes only the `coManaged` and `alsoManaged` files, which are defined in the `dataReg`.

  Note: Some applications have co-managed files stored at different directory levels. These co-managed files will also be included for copy.

  Note: The `prop.xx` file, which is invalid in OpenAccess libraries, is no longer automatically included in co-managed sets.

- `CCP_EXPAND_ALL`: Expands everything in a directory.

q_expandSkipLibList

gdmSpecList object containing the libraries to exclude while copying. *q_expandSkipLibList* is a stop list—design objects in the libraries are not traversed.

For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.

This argument is optional; its default value is `nil`.

l_expandViewTypeList

List of strings specifying the view types to be expanded. If the list is empty, includes all view types in the hierarchy.

This argument is optional; its default value is `nil`.

l_expandViewNameList

List of view names to be expanded. If the list is empty, includesall view names in the hierarchy.

This argument is optional; its default value is `nil`.

t_vNameSimExp

Regular expression specifying the views to expand.

t_vNameSpace	The name space in which the arguments *t_vNameSimExp* and *l_expandViewNameList* are provided. Can be one of the following strings: `VHDL`, `Verilog`, `Verilog-A`, `CDBA`, `Concept`, `Library`, `LibraryUnix`, or `LibraryNT`.. For a list of Cadence name spaces, see Name Spaces for Different Data Types.
g_addProp	`t` or `nil`; default is `nil`. If you specify `t`, library and cell property files are also copied. It is recommended that you use `nil` for this argument because of potential conflicts between property files. Properties are usually added by the copy updaters.
g_existenceCheck	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to validate the existence of layers, purposes, viaDefs, and siteDefs in the technology database of the destination library after the copy command is completed. **Note:** This argument is typically used with copy functions and is not needed for expansion functions.
g_reReferenceCustomVias	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to update custom via definitions to point to cellviews in the destination library. **Note:** This argument is typically used with copy functions and is not needed for expansion functions.

Value Returned

q_expSpecList	A gdmSpecList object containing gdmSpec objects for the files in the expansion list.
`nil`	The configuration could not be expanded.

ccpGetAutoRename

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

Description

Gets the current status of the auto rename feature. Earlier, the status of this feature was controlled only by a single time evaluation of the `CDS_COPY_AUTO_RENAME` shell environment variable. This environment variable defines the initial status setting for this feature. Now, the ccpSetAutoRename function allows you to change the status after the startup initialization.

Arguments

None.

Value Returned

`t`	The auto rename feature is enabled.
`nil`	The auto rename feature is disabled.

Example

```
ccpGetAutoRename()
=> t
```

The auto rename feature is enabled.

ccpSetAutoRename

```
ccpSetAutoRename(
     g_general
     )
     => t
```

Description

Sets the status of the auto rename feature to on or off. Earlier, the status of this feature was controlled only by a single time evaluation of the `CDS_COPY_AUTO_RENAME` shell environment variable. This environment variable defines the initial status setting for this feature. Now, this function allows you to change the status after the startup initialization.

Arguments

`g_general`	`t` or `nil`; If `t` is specified, the auto rename feature is enabled and the `CDS_COPY_AUTO_RENAME` environment variable is set to `YES`. If `nil` is specified, the auto rename feature is disabled and the `CDS_COPY_AUTO_RENAME` environment variable is set to `NO`.

Value Returned

`t`	Sets the current status of the auto rename feature to on or off.

Example

```
; Calling ccpCopy with autorename mode switched to on.
    autoRenameWasOn = ccpGetAutoRename()
    if( !autoRenameWasOn  ccpSetAutoRename(t) )

    ok = ccpCopy( sourceGdmList destinationGdmList overwriteBoolean )
; Optional restoring autorename back to off for this example
    if( !autoRenameWasOn  ccpSetAutoRename(nil) )
```

If the `ccpSetAutoRename` function is set to `t`, the auto rename feature gets enabled. However, when the `ccpSetAutoRename` function is set to `nil`, the auto rename feature gets disabled.

ccpRename

```
ccpRename(
     G_src
     G_dest
     [ g_overWrite ]
     [ s_expFlag ]
     [ s_whatToUpdate ]
     [ q_updateLibList ]
     [ g_existenceCheck ]
     [ g_reReferenceCustomVias ]
     )
     => t / nil
```

Description

Renames a library, cell, view, or file. The source library, cell, view, or file is copied, and then the original one is deleted. When you copy a directory (library, cell, or view), all the files in the directory are copied, regardless of whether they are co-managed or not.

Note: You can set the maximum cell name length allowed by using the `CDS_MAX_CELL_NAME_LENGTH` environment variable.

Arguments

G_src	Source gdmSpec object. Represents the library, cell, view, or file to be renamed. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
G_dest	Destination gdmSpec object. Must be of the same type as the source object. For example, both must be libraries. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
g_overWrite	Boolean overwrite flag. Its value can be `t` or `nil`. (Any other value is considered `t`.) Overwrites the object if it already exists at the destination. If *g_overWrite* is `t` and the object is managed and not already checked out, it is checked out by the system.
s_expFlag	Expand option—one of the following predefined symbols: ■ `CCP_EXPAND_COMANAGED`: Expands a directory and includes only the `coManaged` and `alsoManaged` files, which are defined in the `dataReg`. Illegal if your source gdmSpec object is a file. **Note:** Some applications have co-managed files stored at different directory levels. These co-managed files will also be included for copy. **Note:** The `prop.xx` file, which is invalid in OpenAccess libraries, is no longer automatically included in co-managed sets. **Note:** The `prop.xx` file, which is invalid in OpenAccess libraries, is no longer automatically included in co-managed sets. ■ `CCP_EXPAND_ALL`: Expands everything in a directory. **Note:** The *s_expFlag* argument is currently ignored. All the files in the directory are renamed, regardless of whether you specify `CCP_EXPAND_COMANAGED` or `CCP_EXPAND_ALL`.

s_whatToUpdate	One of the following predefined symbols, which tells the cross-reference updater what to update: ■ `CCP_UPDATE_COPIED_DATA`: Updates cross-references in only the renamed data. ■ `CCP_UPDATE_DESTLIB_ONLY`: Updates all cross-references in the destination library. ■ `CCP_UPDATE_FROM_LIBLIST`: Updates cross-references in the libraries you specify in the next argument, *q_updateLibList*. This argument is optional; its default value is `CCP_UPDATE_COPIED_DATA`.
q_updateLibList	Valid only if the *s_whatToUpdate* argument has a value of `CCP_UPDATE_FROM_LIBLIST`. Otherwise, this argument is ignored. *q_updateLibList* takes a gdmSpecList object containing library gdmSpec objects, which specify the libraries in which to update cross-references. **Note:** If *q_updateLibList* is empty and *s_whatToUpdate* is `CCP_UPDATE_FROM_LIBLIST`, rename proceeds but no updating is done. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
g_existenceCheck	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to validate the existence of layers, purposes, viaDefs, and siteDefs in the technology database of the destination library after the copy command is completed.
g_reReferenceCustomVias	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to update custom via definitions to point to cellviews in the destination library.

Value Returned

`t`	The source object was renamed.
`nil`	The source object could not be renamed.

ccpRenameReferenceLib

```
ccpRenameReferenceLib(
     G_fromLib
     G_toLib
     q_updateList
     [ s_whatCanChange ]
     [ g_existenceCheck ]
     [ g_reReferenceCustomVias ]
     )
     => t / nil
```

Description

Renames, in a library or a subset of a library, references to an external library. The external library is not changed.

Arguments

G_fromLib	gdmSpec object that represents the old library. References to this library will be replaced with the value of *G_toLib*. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
G_toLib	gdmSpec object that represents the new library. References to the library you specify in *G_fromLib* will be replaced with references to this library. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
q_updateList	gdmSpecList object that specifies the library in which references are updated. The value of this argument must be a library, or a list of files, views, or cells that belong to the same library. For information about creating gdmSpec and gdmSpecList objects, see Generic Design Management (GDM) Functions.
s_whatCanChange	One of the following predefined symbols, which tells the cross-reference updater what to update: ■ `CCP_ALL_REFS`: Updates all the references. ■ `CCP_VALID_TO_REFS`: Updates references only if the *G_toLib* library exists. ■ `CCP_VALID_FROM_REFS`: Updates references only if the *G_fromLib* library exists. ■ `CCP_VALID_BOTH_REFS`: Updates references only if both the *G_toLib* and the *G_fromLib* libraries exist. The default value of this argument is `CCP_ALL_REFS`.
g_existenceCheck	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to validate the existence of layers, purposes, viaDefs, and siteDefs in the technology database of the destination library after the copy command is completed.
g_reReferenceCustomVias	`1` or `0` (default). Sets a flag for the Virtuoso post-copy updater to update custom via definitions to point to cellviews in the destination library.

Value Returned

`t`	The references were renamed.
`nil`	The references were not renamed.

ccpRegMonitor

```
ccpRegMonitor(
     g_copyMonitor
     )
     => oldCopyMonitor / nil
```

Description

Registers the copy monitor that you create. A copy monitor is a program that displays the progress of the copy command while it is being executed. For an example of a copy monitor, see the copy monitor used in the Library Manager's Copy Wizard.

To write a copy monitor function, use the format described in Copy Monitor Function Format.

Arguments

g_copyMonitor	The copy monitor that you want to register, specified as a symbol. Registering a copy monitor removes any copy monitor that was previously registered. If you only want to remove the previous copy monitor but do not want to register a new copy monitor, specify `nil` for this argument.

Value Returned

oldCopyMonitor	The previously registered copy monitor, if any.
`nil`	There is no previously registered copy monitor.

Example

```
crm_ret = ccpRegMonitor('copyMonitor)
printf("ccpRegMonitor returned \"%L\"\n" crm_ret)
```

See also "cdsCopy SKILL Examples" on page 142.

Copy Monitor Function Format

If you write a copy monitor function, you must write it in the format described in this section.

Note: The copy monitor function must be fast, otherwise it will affect the performance of the copy command.

The copy monitor must return `nil` for the copy operation to continue; any non-`nil` return value cancels the copy operation.

Use the following format for the copy monitor function that you write:

```
procedure(copyMonitor(myFunction copyPhase fromPath toPath fromSpec toSpec
numCount numTotal "ttttggxx"))
=> nil / non-nil
```

where

- *`myFunction`* is your copy monitor function's symbol.
- *`copyPhase`* is the string `"copy"`.
- *`fromPath`* is the path (specified as a string) of the file that is being copied.

  An empty string is also a legal value.
- *`toPath`* is the path (specified as a string) to which the file is copied.

  An empty string is also a legal value.
- *`fromSpec`* is the gdmSpec object of the file that is being copied.

  `nil` is also a legal value.
- *`toSpec`* is the destination gdmSpec object.

  `nil` is also a legal value.
- *`numCount`* is the count, which is incremented by 1 for each file that is copied.

  The count is terminated at -1: for every copy operation, there is a final call to the copy monitor with `numCount = -1`.
- *`numTotal`* is the total number of files that have to be copied.

  This number may vary during a copy operation, for example, it may increase when the cellview subdirectories are expanded. If this number is negative, it indicates that the copy operation is failing.

Example: Copy Monitor Function

The following example creates a copy monitor function that prints some of its arguments.

```
procedure(copyMonitor(
    copyFn copyPhase fromPath toPath fromSpec toSpec numCount numTotal "ttttggxx")
    if(geqp(numCount 0) then
        if(neq(fromSpec nil) then
            printf("Monitor: %s %L -> %L, %d of %d files.\n"
                        copyFn
                        gdmInspectSpec(fromSpec "CDBA")
                        gdmInspectSpec(toSpec "CDBA")
                        numCount numTotal)
            ))
    nil
)
```

ccpRegTrigger

```
ccpRegTrigger(
     t_copyPhaseStr
     s_triggerFunction
     [ g_canEditList ]
     )
     => t / nil
```

Description

Registers the copy trigger that you create. The copy trigger is called in the copy phase that you specify.

Write the copy trigger function in the format described in Copy Trigger Function Format.

Note: If you run copy commands through the Cadence Library Manager, you need to set the following in your `.cdsenv` file so that copy is done by cdsCopy instead of the Library Manager and the copy triggers that you specify are called:

```
ddserv.lib                   enableCopyInDFII     boolean     t
cdsLibManager.copyGlobals    mpsRadio             toggle      (t nil)
```

(The `cdsLibManager.copyGlobals` setting is the default setting for the *Remote Copy Service* option in the Library Manager's Copy Preferences form.)

Arguments

`t_copyPhaseStr`	One of the following strings, representing the phase of the copy operation in which you want the trigger function to be called: ■ `"ccpPostExpandTrigger"` ■ `"ccpPreTransferTrigger"` ■ `"ccpPostTransferTrigger"` ■ `"ccpPreUpdateTrigger"` ■ `"ccpPostCopyTrigger"`
`s_triggerFunction`	Your trigger function, specified as a symbol.
`g_canEditList`	`t` or `nil` If you specify `t`, the trigger is allowed to edit the copy lists; if you specify `nil`, the trigger cannot edit the copy lists. The performance of the copy operation is better if you specify `nil` for this argument.

Value Returned

`t`	The copy trigger was added.
`nil`	The copy trigger could not be added.

Example

```
ccpRegTrigger("ccpPostExpandTrigger" 'copyTriggerPrint t)
```

See also “cdsCopy SKILL Examples” on page 142.

Copy Trigger Function Format

A copy trigger function is a function that specifies the action to be triggered in a specific phase of the copy operation.

The copy trigger function must return `t` for the copy operation to continue; if the function returns `nil`, the copy operation is canceled.

Use the following format for the copy trigger function:

```
procedure(copyTrigger(myFunction copyPhaseStr checkOffList supplementList
otherFromSpecs otherToSpecs updateList retHint ctxList reserved "stggggggggx"))
=> t / nil
```

where

- *myFunction* is your trigger function's symbol.
- *copyPhaseStr* is the current copy trigger phase, specified as a string.
- *checkOffList* is the list of items to be copied. Each element of the list has the following format:

  `(`*`boolean fromGdmSpec toGdmSpec`*`)`

  If you do not want an item to be copied, set its *boolean* value to `nil`.

  Note: The trigger function can only modify the copy lists if the *g_canEditList* argument of `ccpRegTrigger` is `t`.

  Note: Do not skip any non-derived files because you might lose data. Specifically, if a file is skipped when the trigger is called by a rename function, you will lose the data because the original source will be deleted. The `ccpTagOperation` tag in the *ctxList* argument tells you which function is calling the trigger; you can modify the behavior of the trigger function accordingly.
- *supplementList* is a list to which your trigger function can append additional items to be copied. Any items you add must be in the same format as the items in the *checkOffList*. The *supplementList* ends with a `nil`; if you add items, do not remove the `nil` at the end of the list.
- *otherFromSpecs* and *otherToSpecs* are gdmSpecLists of other items to be copied that cannot be modified by the trigger function.
- *updateList* specifies the items to update. The trigger function can add or remove items from this list. *updateList* contains two elements:
  - *s_updateSet*, which is one of the following symbols:

    ```
    CCP_UPDATE_COPIED_DATA
    CCP_UPDATE_DESTLIB_ONLY
    CCP_UPDATE_FROM_LIBLIST
    ```
  - *q_updateList*, which is a gdmSpecList that contains the list of libraries to update.
- *retHint* is a list of one element which is either `nil` or non-`nil`. If your trigger function modifies any copy list, then it must also set the value of *retHint* to non-`nil`.
- *ctxList* is a list that contains the original arguments from the copy command. Each element of the list has the following format:

`(`*`tag value`*`)`

Your trigger function must not modify these values.

Each tag-value pair is one of the following:

Tag	**Value of type**
`ccpTagSkipLibs`	gdmSpecList
`ccpTagOverwrite`	boolean
`ccpTagExpand`	expansion symbol
`ccpTagViewTypes`	list of strings
`ccpTagViewNames`	list of strings
`ccpTagNameSpace`	string
`ccpTagNameExprs`	string expression
`ccpTagUpdateOpt`	update symbol
`ccpTagUpdateLibs`	gdmSpecList
`ccpTagOperation`	string identifying the copy function that fired the trigger; one of the following: `ccpCopy`, `ccpCopyDesign`, `ccpCopyExactDesign`, `ccpCopyConfig`, `ccpExpand`, `ccpExpandDesign`, `ccpExpandExactDesign`, `ccpExpandConfig`, `ccpRename`, `ccpRenameReferenceLib`

- *`reserved`* is an integer argument that is currently reserved.

Example: Copy Trigger Function

The following example creates a copy trigger function that prints some of its arguments:

```
procedure(copyTriggerPrint(myFunction copyPhaseStr checkOffList supplementList
                           otherFromSpecs otherToSpecs updateList retHint
                           ctxList reserved "stgggggggx")
    let((retOK)
        retOK = t
        printf("Copy phase is \"%s\"\n" copyPhaseStr)
        printf("Calling options were %L\n", ctxList)
        printf("Pre-copy set is %L\n", checkOffList)
        printf("Post-copy is from %L\n", otherFromSpecs)
        printf("                to %L\n", otherToSpecs)
        retOK
    )
)
```

ccpRemoveTrigger

```
ccpRemoveTrigger(
     t_copyPhaseStr
     s_triggerFunction
     )
     => t / nil
```

Description

Removes the registered copy trigger function from the copy phase you specify.

See ccpRegTrigger for more information about copy triggers.

Arguments

t_copyPhaseStr	One of the following strings, representing the phase of the copy operation in which the trigger function was registered: ■ `"ccpPostExpandTrigger"` ■ `"ccpPreTransferTrigger"` ■ `"ccpPostTransferTrigger"` ■ `"ccpPreUpdateTrigger"` ■ `"ccpPostCopyTrigger"`
s_triggerFunction	The registered trigger function, specified as a symbol.

Value Returned

`t`	The copy trigger was removed from the phase you specified.
`nil`	The copy trigger could not be removed.

cdsCopy SKILL Examples

Example 1

The following SKILL `.il` file copies library `myLib` to `myLibCopy`:

```
viewtypeList = '(maskLayout schematicSymbol)
srcList = gdmCreateSpecList()
src = gdmCreateSpec("myLib" "" "" "" "CDBA")
gdmAddSpecToSpecList(src srcList)
destList = gdmCreateSpecList()
dest = gdmCreateSpec("myLibCopy" "" "" "" "CDBA")
gdmAddSpecToSpecList(dest destList)
ccpCopy(srcList destList nil 'CCP_EXPAND_ALL viewtypeList nil)
```

Example 2

The following SKILL `.il` file creates and registers a copy monitor and a copy trigger function and uses them in a copy operation:

```
; An example copy monitor function, which prints some of its arguments
procedure(copyMonitor (
   copyFn copyPhase fromPath toPath fromSpec toSpec numCount numTotal "ttttggxx" )
    if(geqp(numCount 0) then
        if(neq(fromSpec nil) then
            printf("Monitor: %s %L -> %L, %d of %d files.\n"
                        copyFn
                        gdmInspectSpec(fromSpec "CDBA")
                        gdmInspectSpec(toSpec "CDBA")
                        numCount numTotal)
            ))
    nil
)

; An example copy trigger function, which prints some of its arguments
procedure(copyTriggerPrint(myFunction copyPhaseStr checkOffList supplementList
otherFromSpecs otherToSpecs updateList retHint ctxList reserved "stgggggggx")
    let((retOK)
        retOK = t
        printf("Copy phase is '%s'\n" copyPhaseStr)
        printf("Calling options were %L\n", ctxList)
        printf("Pre  copy set is %L\n", checkOffList)
        printf("Post copy is from %L\n", otherFromSpecs)
        printf("               to %L\n", otherToSpecs)
        retOK
    )
)

; Registering the copy monitor and copy trigger functions
crm_ret = ccpRegMonitor('copyMonitor)
printf ("ccpRegMonitor('symbol) returned '%L'\n" crm_ret)

ccpRegTrigger("ccpPostExpandTrigger" 'copyTriggerPrint nil)
ccpRegTrigger ("ccpPostCopyTrigger" 'copyTriggerPrint nil)

; Copy
testLib = "MonitorTestLib"
srcList = gdmCreateSpecList()
src = gdmCreateSpec("analogLib" "cap" "" "" "CDBA" 0)
gdmAddSpecToSpecList(src srcList)

destList = gdmCreateSpecList()
dest = gdmCreateSpec(testLib "" "" "" "CDBA" 0)
gdmAddSpecToSpecList(dest destList)

updList = gdmCreateSpecList()
upd = gdmCreateSpec(testLib "" "" "" "CDBA" 0)
gdmAddSpecToSpecList(upd updList)
upd = gdmCreateSpec("CadenceBasic" "" "" "" "CDBA" 0)
gdmAddSpecToSpecList(upd updList)

tlib = ddGetObj(testLib)
vnList = '(schematic cmos_sch symbol)
vtList = '(ComposerSymbol ComposerSchematic)

if(neq(tlib nil) then
    printf("Preparing by deleting test library '%s'\n" testLib)
    ddDeleteObj(tlib))
ccpCopy(srcList destList nil 'CCP_EXPAND_ALL vtList vnList "s* v*" "CDBA"
'CCP_UPDATE_FROM_LIBLIST destList)

printf("Cleaning up by deleting test library '%s'\n" testLib)
ddDeleteObj(ddGetObj(testLib))
```

4

Library Manager Functions

This chapter describes the following SKILL functions as indicated in the *Used in* column:

Function	Used in	Description
`lmgrAddMenuItems`	`cdsLibMgr.il` (only)	Appends objects to the specified menu
`lmgrCreateMenu`	`cdsLibMgr.il` (only)	Creates a structure for a menu
`lmgrCreateMenuItem`	`cdsLibMgr.il` (only)	Creates an instance of a menu item
`lmgrDefineInits`	`cdsLibMgr.il` (only)	Defines the initialization action to perform when the customization code begins and the termination action to perform immediately before the Library Manager terminates
`lmgrDeleteMenuItems`	`cdsLibMgr.il` (only)	Deletes objects from the menu/ pop-up hierarchy
`lmgrDisplayMessage`	Virtuoso (only)	Displays a string in the Library Manager message window an dappends it to the Library Manager log file
`lmgrGetObject`	`cdsLibMgr.il` Virtuoso	Returns a list describing the state of a single named object
`lmgrInsertMenuItems`	`cdsLibMgr.il` (only)	Inserts objects into the menu
`lmgrManageMenuItems`	`cdsLibMgr.il` Virtuoso	Manages the named objects, enabling the display of these objects inside their menu hierarchy and pop-up sets

Function	Used in	Description
`lmgrMenuSubsInPopup`	`cdsLibMgr.il`	Places the specified menu in the sub-pop-up menu, which appears when you right-click in the Library Manager window.
`lmgrQueryNamedObjects`	Virtuoso (only)	Returns a list containing the names of all the menu items on the Library Manager menu
`lmgrSensitizeMenuItems`	`cdsLibMgr.il` Virtuoso	Sensitizes the named objects, enabling the active state of these objects inside their menu hierarchy and pop-up sets
`lmgrSetObject`	`cdsLibMgr.il` Virtuoso	Describes the state of the object
`lmgrVerbose`	Virtuoso (only)	Specifies the amount of information displayed in the CIW

lmgrAddMenuItems

```
lmgrAddMenuItems(
      t_menuName
      t_popupSet
      l_names
      )
      => t / nil
```

Description

Appends the named objects to the specified menu. An item can be another menu. Each menu item or menu can be added to only one menu. Likewise, any menu item can appear at most once in every pop-up menu. Menus cannot be added to a pop-up. Pop-ups must remain a flat structure only. Pop-ups cannot be added to menus.

You can add only menus to the main pull-down menu bar. Do this by specifying an empty string "", or the reserved name `menuBar` for the `t_menuName` argument. It is also legal to specify the `t_menuName` argument as `popup` or `nil` when the sole purpose is to add an already added object to an indicated pop-up set.

If radio items are added to a particular menu, then only radio items can be added to that menu. The pull-down menu or submenu cannot contain both radio and non-radio items.

Arguments

t_menuName	The name of the menu, to which items are added.
t_popupSet	A string of characters indicating which pop-ups this item is also to be included in.The pop-up set is a string including 0 or more characters using the following mapping: `L` = included in library list `C` = included in cell list `V` = included in view list `l` = included in library file list `c` = included in cell file list `v` = included in view file list `t` = included in category list
l_names	List of item names that indicates items to be added.

Value Returned

`t`	All arguments were processed without error.
`nil`	An error occurred, You might have added a menu item to more than one menu or added a menu item that does not exist.

Example

```
; Add into the new pull-down menu and into
; the Library and Cell popups
lmgrAddMenuItems ( "MyPulldown" "LC" '( "myMenuItem1" ) ) => t

; Add the new pull-down menu into the menu banner
lmgrAddMenuItems ( "menuBar" "" '( "MyPulldown" ) ) => t
```

Reference

`lmgrCreateMenu, lmgrCreateMenuItem, lmgrInsertMenuItems`

lmgrCreateMenu

```
lmgrCreateMenu(
     t_menuName
     l_menuAttributes
     )
     => t / nil
```

Description

Creates a structure for a menu, which can be populated with menu items. Once this menu is created, you can install it into the top menu bar or into another parent menu using `lmgrAddMenuItems`.

You can specify a predefined menu name in order to change its label or to add a single map callback attribute to a predefined menu. See Callback Definition List in the *Cadence Library Manager User Guide* for details on how to specify a callback list.

A map callback must finish its execution quickly. Lengthy callback options are automatically ignored. Additionally, the Library Manager may choose to not wait for map callbacks to finish if callbacks require more than 5 seconds elapsed time. The reason behind these restrictions is that the X Window System remains locked for all other programs while a map callback is being processed. Efficient shared usage necessitates fast and efficient execution of map callbacks.

The predefined name `pop-up` refers to all pop-ups started from the main selection lists. The `label` and `font` attributes have no significance within a pop-up menu. This mechanism exists only to allow specifying the map callback of pop-ups.

Arguments

`t_menuName`	A unique global name to reference this menu.

`l_menuAttributes`	List of tagged elements consisting of name-value pairs for the following attributes:
	`label`
	Paired value is the string displayed with this menu. This attribute is mandatory when specifying a custom (non predefined) menu.
	`font`
	Paired value is a string for the font to use for this particular menu. If this attribute is not specified, or if `nil` or an empty string is specified, the global default fonts are used.
	`mapCallback`
	Value is the callback list defining the action, if any, which will be called directly before mapping the menu.

Value Returned

`t`	All arguments were processed without error.
`nil`	An error occurred. In this case, the menu will *not* be created.

Example

```
lmgrCreateMenu ( "MyPulldown" '( ( "label" "My pulldown" ) ) ) => t
```

Creates a new pulldown menu.

Reference

`lmgrCreateMenuItem, lmgrAddMenuItems, lmgrInsertMenuItems`

lmgrCreateMenuItem

```
lmgrCreateMenuItem(
      t_itemName
      t_itemType
      l_itemAttributes
      )
      => t / nil
```

Description

Creates an instance of a menu item, which should be populated into a single menu. Once this menu item is created, you can install it into a menu using `lmgrAddMenuItems()`.

There are four types of menu items: `simple`, `toggle`, `radio`, and `separator`. `simple` is a normal menu item, `toggle` is a menu item with a tick box next to it, `separator` is a line to draw between menu items. You define one radio item for each choice in the set of radio options. All the radio items in the set must be collected in their own pull-down menu or submenu.

See Callback Definition List in the *Cadence Library Manager User Guide* for details on how to specify the callback list.

Arguments

t_itemName	A unique global name to reference this menu item.
t_itemType	A standard type, one of `simple`, `toggle`, `radio`, `separator`.
l_itemAttributes	List of tagged elements consisting of name value pairs for the following attributes. None of these attributes are valid for separators:

- `label`: Paired value is the string displayed by this item. This attribute is mandatory except when the item type is `separator`.
- `mnemonic:` Paired value is a string representing the character mnemonic to use with this object.
- `accelerator`: This attribute is not supported in the current release.
- `font:` Paired value is a string for the font to use for this particular item. If this attribute is not specified, or if *nil* or an empty string is specified, the global default fonts are used.
- `callback`: Paired value is the callback list defining the action, if any, and the interpretation of results, which will be called when activating this menu item. See the previous section describing callback definitions. For radio items, when the item is selected through the user interface, the callback for both the item being selected and the item being deselected is started.

Value Returned

`t`	All arguments were processed without error.
`nil`	An error occurred. In this case, the menu item will not be created.

Example

```
lmgrCreateMenuItem ( "myMenuItem1" "simple"
    '( ( "label" "Do My Thing" )
      ( "callback" ( "myThingCallback" ) )
      )
    )              => t
```

Reference

`lmgrCreateMenu`, `lmgrAddMenuItems`, `lmgrInsertMenuItems`

lmgrDefineInits

```
lmgrDefineInits(
     l_initCallback
     l_termCallback
     )
     => t / nil
```

Description

Defines the initialization action to perform when the customization code begins (after parsing the extension file) and the termination action to perform immediately before Library Manager terminates. You must define the callback in the Virtuoso program before it starts. Otherwise, you get an error. No arguments are passed to either callback.

Arguments

l_initCallback	A callback definition using the syntax described earlier.
l_termCallback	A callback definition using the syntax described earlier.

Value Returned

`t`	All arguments were processed without error.
`nil`	An error occurred.

Example

```
lmgrDefineInits( '( "myInitLibMgr" ) '( "myCloseLibMgr" ) )
=> t
```

Defines a callback to be started when Library Manager is started and another when it terminates. This can then do some dynamic customization, depending on the current session.

lmgrDeleteMenuItems

```
lmgrDeleteMenuItems(
     l_names
     )
     => t / nil
```

Description

Deletes the named objects from the menu/pop-up hierarchy. The object can be either a single item or an entire menu. There is no recovery or undo from this operation. If you merely wish to temporarily disable the view of an object, consider using `lmgrManageMenuItems()` instead.

You can also specify items that have been predefined by Cadence. This function is supplied in the Library Manager local extension file parse environment so you can use it predefined objects. In most cases, newly defined custom items are not deleted. Otherwise, there is no reason to create them in the first place.

Arguments

l_names	List of item names to be deleted.

Value Returned

`t`	All arguments were processed without error.
`nil`	An error occurred.

Example

```
lmgrDeleteMenuItems( "designCascade" )
=> t
```

Permanently removes the *Design Manager* menu and all associated menu items. It also removes the menu items from the pop-up sets.

Reference

`lmgrManageMenuItems, lmgrSensitizeMenuItems`

lmgrDisplayMessage

```
lmgrDisplayMessage(
     t_text
     [ g_logOnly ]
     )
     => t / nil
```

Description

Displays a string in the *Message* area on the Library Manager form and appends it to the Library Manager log file. If desired, the message can be written just to the Library Manager log file and not echoed to the user. Ensure that the string is terminated by a newline character, or the string might be confused with the output from other Library Manager commands or other calls to this function.

Arguments

t_text	The string to output in the Library Manager log file.
g_logOnly	An optional Boolean that echoes the string to the message window and writes it to the log file if the string is not specified or is specified as `nil`, and writes the string just to the log file otherwise.

Value Returned

`t`	The string is successfully transmitted to the Library Manager.
`nil`	Either no Library Manager process is running or an error occurred in transmitting the string.

Example

```
lmgrDisplayMessage( "Virtuoso talking to libManager" )
=> t
```

This example writes a short message to the log file.

Reference

`lmgrVerbose`

lmgrGetObject

```
lmgrGetObject(
     t_objName
     )
     => l_attrList / nil
```

Description

Returns a list describing the state of the single named object or `nil` if the object does not exist. The format of the attribute list will be variable length list of tags (strings) paired with matching values that are either a string or Boolean value.

Arguments

`t_objName`	Name of a menu or menu item.

Value Returned

`l_attrList`	An associated list of tags (strings) paired with values to reflect the status of the named object. The example below shows how this associated list appears. The possible tags are `parent`, `type`, `managed`, `sensitive`, `state`, `label`, `mnemonic`, and `font`. The `state` value can be `nil` for objects that have no state, such as for separators or simple buttons. The state uses `nil` for `off/false`, and `t` for `on/true`. This also applies to the Boolean `managed` and `sensitive` values. The `sensitive` value returned is the logical AND combination of the current value for the object and the values of the menus. For example, menu `M` contains menu item `I`. `M` is not sensitive, but `I` is sensitive, so the `sensitive` value returned for `I` would be `nil`.
`nil`	The named object does not exist.

Example

```
lmgrGetObject( "viewFilterButton" ) =>

(( "parent" "viewCascade" )
    ( "type" "simple" )
    ( "managed" t )
    ( "sensitive" t )
    ( "label" "Filters... " )
    ( "font" nil )
    ( "mnemonic" nil )
)
```

Retrieves the settings of the *View – Filters* menu item.

Reference

`lmgrSetObject`

lmgrInsertMenuItems

```
lmgrInsertMenuItems(
     t_menuItem
     t_popupSet
     l_names
     )
     => t / nil
```

Description

Inserts the named objects into the menudirectly before the indicated menu item. An object can be an entire menu. Each menu item or menu can be inserted or added to only one menu. Likewise, any menu item can appear at most once in each pop-up menu. Menus can not be inserted (added) to a pop-up. Pop-ups must remain a flat structure only. You cannot insert or add pop-ups to menus. This function is similar to `lmgrAddMenuItems()`, except in the meaning of the first argument.

Only menus can be added to the main pull-down menubar. This is indicated by specifying an empty string "" or the reserved name `menuBar` for the `t_menuName` argument. It is also legal to specify the `t_menuItem` argument as `pop-up` or `nil` when the sole purpose is to add an already added user-defined object to an indicated pop-up set.

If radio items are added to a particular menu, then only radio items may be added to that menu. The pull-down menu or submenu may not contain both radio and non-radio items.

Arguments

t_menuName	The name of the menu to which the items are added.
t_popupSet	A string of characters indicating which pop-ups this item is to be included in. The pop-up set is a string including 0 or more characters using the following mapping: ■ `L` = included in library list ■ `C` = included in cell list ■ `V` = included in view list ■ `l` = included in library file list ■ `c` = included in cell file list ■ `v` = included in view file list ■ `t` = included in category list
l_names	List of item names that indicates items to be added.

Value Returned

`t`	All arguments were processed without error.
`nil`	An error occurred. You might have added a menu item to more than one menu or added a menu item that does not exist.

Example

```
; Add into the Edit pull-down menu, before the Copy item
; and not into any popups
lmgrInsertMenuItems ( "copySimpleButton" "" '( "myMenuItem2" ) ) => t
```

Reference

`lmgrCreateMenu, lmgrCreateMenuItem, lmgrAddMenuItems`

lmgrLogShowPopup

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

Description

Restores the context for popup menus for Library Manager replay. This function is only intended for supporting replay and cannot be called from other SKILL engine programs. You do not need to use this function directly.

lmgrManageMenuItems

```
lmgrManageMenuItems(
     l_names
     g_manageOn
     )
     => t / nil
```

Description

Manages the named objects (enabling the display of named objects) inside their menu hierarchy and pop-up sets. Objects can be either a single item or an entire menu.

You can also specify items that have been predefined by Cadence.

This function has the same effect as specifying the `managed` attribute in `lmgrSetObject()`.

Arguments

`l_names`	List of item names that indicates items to be deleted.
`g_manageOn`	A Boolean that sets unmanaged (invisible) if it is `nil`, and managed otherwise.

Value Returned

`t`	All arguments were processed without error because the named objects exist.
`nil`	An error occurred.

Example

```
lmgrManageMenuItems( list ( "fileSeparator1" "fileSeparator2"
"fileSeparator3" "fileSeparator4" ) nil ) => t
```

This example causes the separators on the *File* menu to be made invisible.

Reference

`lmgrDeleteMenuItems, lmgrSensitizeMenuItems`

lmgrMenuSubsInPopup

```
lmgrMenuSubsInPopup(
     t_menuName
     g_isSubMenu
     )
     => t / nil
```

Description

This API is used only in the Library Manager's `cdsLibMgr.il` file for menu customization. The named menu may be placed into a sub-menu of a popup menu, which appears on clicking the right mouse button (RMB) in the Library Manager's window that lists the library, cell, and view items being browsed. This will be placed in a sub-popup if the boolean value was set to be `true`. In addition, menu items tagged for adding to the popup menus are displayed within its menu's sub-popup (if any). Sub-popup menus may be hierarchically nested per menu as they have been setup/defined through the customized menus' hierarchy.

Arguments

`t_menuName`	String for menu's name.
`g_isSubMenu`	Default value is `t` or `nil`. `t` sets the menu to be shown in the popup as a sub-menu within the popup. `nil` means the menu will not be displayed, and its items will be "flattened" into its owner's level, i.e. the previous behavior.

Value Returned

`t`	The previous value set for the sub-menu attribute.
`nil`	An invalid menu name was specified.

Example

```
lmgrMenuSubsInPopup( "designCascade" t)
=> t
```

In this example the menu name is "designCascade". This will place the Cadence standard and topmost "Design Manager" (labeled) menu's items into a sub-menu for the RMB popup.

lmgrQueryNamedObjects

```
lmgrQueryNamedObjects(
     [ t_menuName ]
     )
     => l_list / nil
```

Description

Returns a list containing the names of all menu items in the Library Manager menu named *`t_menuName`*.

If *`t_menuName`* is not specified, the names of all menus and menu items in the Library Manager are returned. The order in the list is arbitrary. The list will contain the names of any standard Library Manager objects as well as any objects which have been specified in the `cdsLibMgr.il` file.

Arguments

`t_menuName`	The option menu name to query.

Value Returned

`l_list`	The list of menu names found.
`nil`	Either no Library Manager process is running or an error occurred in retrieving the information, for example, a nonexistent menu name was specified.

Example

```
lmgrQueryNamedObjects( "viewCascade" )
=>( "viewFilterButton" "viewSeparator1" "viewRefreshButton" )
```

Retrieves the menu items for the *View* menu.

lmgrSensitizeMenuItems

```
lmgrSensitizeMenuItems(
     l_names
     g_sensitive
     )
     => t / nil
```

Description

Sensitizes the named objects (enabling the active state of named objects) inside their menu hierarchy and pop-up sets. Objects can be either a single item or an entire menu.

You can also specify items that have been predefined by Cadence.

This function has the same effect as specifying the `sensitive` attribute in `lmgrSetObject()`.

Caution

Some Cadence predefined "simple" items can become sensitive or insensitive dynamically whenever a selection is made.

For example, the *Check In* and *Check Out* menu items are only sensitive when over a library under design management. Consequently, you might want to permanently disable these menu items by making them unmanaged instead, to avoid your customization being overridden automatically by the Library Manager itself.

Arguments

l_names	List of item names to be deleted.
g_sensitive	If `nil`, the menu or menu item is shown grayed-out so that it cannot be selected. Otherwise, it is drawn as normal and can be selected.

Value Returned

`t`	All arguments were processed without error because the named objects exist.
`nil`	An error occurred.

Example

```
lmgrSensitizeMenuItems( list ( "viewFilterButton" ) nil ) => t
```

This example causes the *View – Filter* menu item to be grayed-out.

Reference

`lmgrDeleteMenuItems, lmgrSensitizeMenuItems`

lmgrSetObject

```
lmgrSetObject(
     t_objName
     l_attrList
     )
     => t / nil
```

Description

Accepts a list describing the state of the named object. Returns `nil` if the object does not exist. The format of the attribute list is a variable-length list of tags (strings) paired with matching string or Boolean values.

Arguments

`t_objName`	Name of a menu or menu item.
`l_attrList`	Associated list of attribute names and paired values, such as the following:

```
("managed"        t)
("sensitive"      t)
("label"          "Copy")
("mnemonic"       "C")
("state"        t)
```

Valid attributes to change are `managed`, `sensitive`, `label`, `mnemonic`, `font`, and `state`. Additionally, the following attributes are recognized, but are silently ignored: `accelerator`, `parent`, and `type`. Any other attribute specifications are illegal and generate an error.

Not all attribute values need to be specified in *`l_attrList`*. Unspecified values are left unchanged.

Value Returned

`t`	The function was called correctly.
`nil`	The named object does not exist, or invalid attributes are specified.

Example

```
lmgrSetObject ( "designCascade" '( ( "font" "-adobe-courier-bold-o-
normal--25-180-100-100-m-150-iso8859-1" ) ) ) => t
```

Sets the font of the Design Manager pull-down to a large bold courier font.

Reference

`lmgrGetObject`

lmgrVerbose

```
lmgrVerbose (
     x_level
     )
     => x_level / nil
```

Description

To aid in debugging the callbacks registered in the `cdsLibMgr.il` file, it is possible to get additional informational messages to be output to the CIW.

Arguments

x_level	An integer representing the verbose level to use. `0` means no messages are written and `1` means that messages are written to the CIW. Any other value is reserved for future use.

Value Returned

x_level	The verbose level if a valid value was specified.
`nil`	An invalid verbose level was specified.

Example

```
lmgrVerbose ( 1 ) => 1
```

```
( lmgr ) Calling: abCopyEnableCallback ( "popup_L" "andrew" "" "" "" "" )
( lmgr ) Calling: abRadioExample ( "RadioOption1" "andrew" "" "" "" "" )
( lmgr ) Calling: abRadioExample ( "RadioOption2" "andrew" "" "" "" "" )
( lmgr ) Calling: abShowAllCheckOuts ( "ShowAllCheckOuts" "andrew" "" """" "" )
```

The above example shows the effect of selecting several menus after setting verbose to 1.

Customization Examples Using SKILL

Each of these examples demonstrates some aspect of the Library Manager customization. Some examples solve a particular requirement. Others just demonstrate a principle.

Each example includes both the code you add to the `cdsLibMgr.il` file and the functions you need to define in the Virtuoso design environment .

Example 1: Checking for Startup of Virtuoso in a Design Management System

In this example, the initialization procedure is used to check whether Virtuoso design environment has been started from a design management workarea. If not, the *Design Manager* pull-down menu and all of the menu items contained within it are set to be unmanaged. This simplifies the appearance of the Library Manager menus for those not using design management.

1. Add the following code to the `cdsLibMgr.il` file:

```
; specify names of functions to call at startup and exit
lmgrDefineInits('("abLibMgrInitProc") '("abLibMgrCloseProc"))
```

2. Define the following functions in Virtuoso, such as in the `.cdsinit` file:

```
/******************************************************************
*                                                                 *
*                       abLibMgrInitProc()                        *
*                                                                 *
*   When Library Manager is initialized, this gets invoked.    *
* It checks to see if we're in a DM workarea, and if not, removes *
*             all the DM menus from the libManager.               *
*                                                                 *
******************************************************************/

procedure ( abLibMgrInitProc ( )
    let ( ( path dm filePD )
        path = simplifyFilename ( getWorkingDir ( ) )
        dm   = ddGetPathDMSys ( path )
        ; if you're not in a DM workarea, then remove the Design
Manager
        ; menu
        when ( null ( dm ) || equal ( dm "none" )
            lmgrManageMenuItems ( list ( "designCascade" ) nil )
            lmgrDisplayMessage ( "Not in DM workarea, so DM menus
removed\n" )
        )
        t
    )
)

/***************************************************************
*                                                              *
*                     abLibMgrCloseProc ()                      *
*                                                              *
*                   Sample close procedure.                   *
*                                                              *
***************************************************************/

procedure ( abLibMgrCloseProc ( )
    ; no reason to display lib manager, since it has been closed.
    printf ( "My closing procedure was called\n" )
    )
```

Example 2: Displaying Checked-Out Library Manager Cellviews and Files

This example uses a script, `tdmfindallcheckouts`, to display all files and cellviews in a library selected within the Library Manager that are checked out and opened by any user.

The menu command starts the callback in Virtuoso, which then uses asynchronous techniques to start the script that uses SKILL IPC. Because the program does not block the user interface, you do not need to wait for it to finish. When the results are available, they are postprocessed and displayed in a Show File window.

This example demonstrates the principles of using an external program via IPC and of displaying the results to the user using a Virtuoso applet.

1. Add the following code to the `cdsLibMgr.il` file:

```
lmgrCreateMenuItem ( "ShowAllCheckOuts" "simple"
    '( ( "label" "Show All CheckOuts" )
    ( "callback" ("abShowAllCheckOuts" ) )
    )
    )

; show the pop-up in library (L) list box only, and on the
; design manager pull-down menu
lmgrAddMenuItems ( "designCascade" "L" '( "ShowAllCheckOuts" ) )
```

2. Define the following code in the Virtuoso session:

```
; define the command to execute to find out the checkout information
unless ( boundp( 'abShowAllCheckOutsCommand )
    abShowAllCheckOutsCommand="tdmfindallcheckouts -lisp"
    )

* abListCompare ( a b )                      *
*                                                            *
* Compares two lists of strings, for use in a sort function. *
*                                                            *
*                                                            *
**************************************************************/

procedure ( abListCompare ( a b )
    let( ( ( status 0 ) )
        while ( status==0 && a && b
            status=strcmp ( car ( a ) car( b ) )
            a=cdr ( a )
            b=cdr ( b )
            )
        status<0
        ))

/*******************************************************************
*                                                                      *
*             abShowAllCheckOutsPostHandler(childexitStatus) *
*
* Read the results of tdmfindallcheckouts - which is in LISP format
* for parsing. Then sort it and display it neatly in a window to
* the user. This function gets called when the tdmfindallcheckouts *
* command completes.
*                                                                    *
*******************************************************************/

procedure( abShowAllCheckOutsPostHandler(child exitStatus)
    let( (fileName filePort readData window owner libName cellName viewName)
        fileName=abShowAllCheckOutsFileTable[child]
        if( fileName then
            /* read the data from the tdmfindallcheckouts output file -
            - it's in LISP syntax, so simple to read */
            filePort=infile(fileName)
            when(filePort
                readData=car(lineread(filePort))
                close(filePort)
                )
            /* remove the file - don't want to leave junk around */
            deleteFile(fileName)
            /* remove entry from the file table */
            remove(child abShowAllCheckOutsFileTable)
            if( readData then
                /* create a text window */
                window=hiCreateWindow(list(100:100 500:400) "text"
"Show File")
                /* put the show file menus in */
                hiInsertBannerMenu(window fileMenu 0)
                /* sort the data, user, then lib, cell, view */
                readData=sort(readData 'abListCompare)
                /* put the titles in the window */
                hiTextDisplayString(window sprintf(nil "%-10s %-10s
%-10s %-10s\n"
                    "Library" "Cell" "View" "CheckOut") nil)
                hiTextDisplayString(window sprintf(nil "%-10s %-10s
%-10s %-10s\n"
                    "=======" "====" "====" "========") nil)
                /* output the entries into the window */
                foreach( entry readData
                    /* set the variables to the values in the list -

                        do it this way to make sure that each
variable has a default value of "" if the list isn't long enough */
                    owner=libName=cellName=viewName=""
                    foreach( (val varName) entry '(owner libName
cellName viewName)
                        set(varName val)
                        )
                    /* display the formatted output in the window */
                    hiTextDisplayString(window
                        sprintf(nil "%-10s %-10s %-10s %-10s\n"
                            libName cellName viewName owner) nil)
                    ) ; foreach
                /* set the title */
                hiSetWindowName(window "Show All Checkouts")
                /* finally, display the window to the user */
                hiDisplayWindow(window)
            else
                lmgrDisplayMessage("No cellViews checked out\n")
            ) ; if
        else
            lmgrDisplayMessage("Couldn't find fileName for show all
checkouts\n")
        ) ; if
        t
    ))

/****************************************************************
*                                                               *
*       abShowAllCheckOuts(MenuName lib cell view file cat)     *
*                                                               *
*     Callback function for show all checkouts. Checks to see   *
*      if a library has been selected, is under DM, and then    *
*      starts the script to find all the checkouts. The post   *
*   handler will take care of reading the results and displaying   *
* them in a window. This means that it will behave asynchronously. *
*                                                               *
****************************************************************/

procedure( abShowAllCheckOuts(MenuName lib cell view file cat)
    let( (fileName dirName handle)
        cond(
            (lib==""
                lmgrDisplayMessage("Must have library name
selected\n")
                )
            (ddAmUsingDM(ddGetObj(lib))
                dirName=ddGetObj(lib)~>readPath
                unless( boundp( 'abShowAllCheckOutsFileTable )
                    abShowAllCheckOutsFileTable=makeTable(
                        'abShowAllCheckOutsTable nil)
                    )
                fileName=makeTempFileName("/tmp/abShowAllCheckOuts")
                ; fire off background process
                handle=ipcBeginProcess(
                    sprintf(nil "%s %s > %s"
                        abShowAllCheckOutsCommand
                        dirName fileName)
                    "" nil nil 'abShowAllCheckOutsPostHandler)
                ; remember where the output was stored
                abShowAllCheckOutsFileTable[handle]=fileName
                lmgrDisplayMessage(
                    sprintf(nil
                        "Finding all checkouts for %s - this may
take some time\n" lib))
                        )
            (t
                lmgrDisplayMessage(sprintf(nil "%s is an unmanaged
library\n" lib))
                )
            ) ; cond
        t
    ))
```

3. Put the script `tdmfindallcheckouts` (shown here) in the UNIX search path:

```
#!/bin/csh -f
#
# Author     John Doe
# Group      Custom IC, Cadence UK
# Machine    SUN
# Date       Mar 13, 1998
# Modified   Mar 26, 1998
# By
#
# script to locate all checkouts from a specified library directories
```

```
#
# Usage: tdmfindallcheckouts [-lisp] pathToLibrary [pathToLibrary…]
#
# SCCS Info: @(#) tdmfindallcheckouts 03/26/98.14:50:20 1.2
#

set progName=$0
set progName=$progName:t

#check for -lisp argument
if ("arg$1" == "arg-lisp") then
    set lispMode
    shift
endif

# check sufficient arguments
if ($#argv < 1) then
    echo "Usage: $progName [-lisp] pathToLibrary"
    exit 1
endif

# choose awk in a platform independent way
if (-x /bin/nawk) then
    set awk=/bin/nawk
else
    set awk=/bin/awk
endif

# if the -lisp argument was specified, write out the query in a form
# that's asily readable from SKILL
if ($?lispMode) then
    tdmls -c $* | $awk '\
BEGIN {printf("(\n")}\
{\
owner=$4;\
paths[1]=paths[2]=paths[3]="";\
numf=split($1,paths,"/");\
paths[numf+1]=$2;\
if(paths[1]!=lib || paths[2]!=cell || paths[3]!=view) \
  printf("(\"%s\" \"%s\" \"%s\" \"%s\")\n", \
    owner,paths[1],paths[2],paths[3]);\
lib=paths[1];cell=paths[2];view=paths[3]}\
END {printf(")\n")}'
else
# otherwise write it out in a human readable form
    echo "The following cellViews and files are checked out by:"
    tdmls -c $* | $awk '{owner=$4;\
paths[1]=paths[2]=paths[3]="";\
numf=split($1,paths,"/");\
paths[numf+1]=$2;\
if(numf<3 || paths[1]!=lib || paths[2]!=cell || paths[3]!=view) \
  print owner ":",paths[1],paths[2],paths[3];\
lib=paths[1];cell=paths[2];view=paths[3]}' | sort
    echo "$progName completed"
endif
```

Example 3: Displaying Cellviews and Files to Be Updated with the Next Update Operation

Use this example to add a new command to display any cellviews and files to be updated when the next update operation is performed.

This example also uses the SKILL IPC to start `tdmupdate -n` on the directory that has been selected in the Library Manager. That is, if a cell is selected, just the updates within that cell will be reported. Rather than displaying the results in a Show File window, the results from `tdmupdate -n` are echoed back into the Library Manager output pane.

1. Add the following code to the `cdsLibMgr.il` file:

```
lmgrCreateMenuItem("ShowUpdates" "simple"
    '(("label" "Show Updates")
      ("callback" ("abShowWhatWillBeUpdated"))
      )
    )

; show the pop-up in library (L), cell (C) and View (V) list boxes
lmgrAddMenuItems("designCascade" "LCV" '("ShowUpdates"))
```

2. Define the following code in the Virtuoso session:

```
/****************************************************************
*                                                               *
*             abTdmUpdateDataHandler(child data)                *
*                                                               *
* Data handler for tdmupdate to echo output into the libManager *
*                        output pane.                           *
*                                                               *
****************************************************************/

procedure(abTdmUpdateDataHandler(child data)
  lmgrDisplayMessage(data)
  )

/***************************************************************
*                                                              *
*            abTdmUpdateExitHandler(child status)              *
*                                                              *
*  When tdmupdate -n exits, put a message into the libManager  *
*                        output pane.                          *
*                                                              *
***************************************************************/

procedure(abTdmUpdateExitHandler(child status)
  lmgrDisplayMessage("tdmupdate -n completed\n")
  )

/***************************************************************
*                                                              *
*          abRunTdmUpdateCheck(@optional fileName)             *
*                                                              *
* Start tdmupdate -check, with a couple of handler functions  *
*                                                              *
***************************************************************/
```

```
procedure( abRunTdmUpdateCheck(@optional fileName)
  let( (tdmUpdateCmd)
    tdmUpdateCmd=if(fileName (strcat "tdmupdate -n " fileName)
                          "tdmupdate -n")
    ipcBeginProcess(tdmUpdateCmd "" 'abTdmUpdateDataHandler nil
           'abTdmUpdateExitHandler)
    )
  )

/***************************************************************
*                                                               *
*        abShowWhatWillBeUpdated(MenuName lib cell view file cat)        *
*                                                               *
* This will cause tdmupdate -n to be started on either the lib/cell/
* view selected, or on the whole workarea if nothing is selected. It
* uses IPC to start the command, and the results will be displayed *
* back into the libManager output pane.
*                                                               *
***************************************************************/

procedure(abShowWhatWillBeUpdated(MenuName lib cell view file cat)
  let((ddObj ddGetObjArgs)
    ddGetObjArgs=setof(lcv list(lib cell view) lcv!="")
    if(ddGetObjArgs
      then
        ddObj=apply('ddGetObj setof(lcv list(lib cell view) lcv!=""))
        lmgrDisplayMessage(
            sprintf(nil
           "The following files will be updated by an update operation
for %s:\n"
                buildString(ddGetObjArgs)))
        abRunTdmUpdateCheck(ddObj~>readPath)
      else
        lmgrDisplayMessage("The following files will be updated by an
update for the whole workarea:\n")
        abRunTdmUpdateCheck()
      )
    t
    ))
```

Example 4: Replacement of File – New – Library in the Library Manager

One potential problem with *File – New – Library* in the Library Manager is that it creates the library itself and then triggers Virtuoso to compile a technology file if needed. Therefore, if you define a `CreateObj` trigger in Virtuoso, the trigger will not be started if the library is created from the Library Manager.

You can resolve this problem by removing the existing facility in the Library Manager and creating a new menu item for which the callback is the function that is started from *File – New – Library* in the CIW.

1. Add the following code to the `cdsLibMgr.il` file:

```
; create a new menu item to start the new library in Virtuoso
; directly, without the lib manager version of the form appearing
lmgrCreateMenuItem("abNewLib" "simple"
    '(("label" "Library ...")
      ("callback" ("abCreateNewLib"))
      ))

; Stop the existing new library "button" from being displayed.
; Do this by unmanaging it
lmgrManageMenuItems(list("FileNewLibButton") nil)
; insert the new menu item before the old, now unmanaged, button.
lmgrInsertMenuItems("FileNewLibButton" "" '("abNewLib"))
```

2. Use the following function in the Virtuoso session:

```
/*******************************************************************
*                                                                  *
*             abCreateNewLib(menuName _l _c _v _f _cat)            *
*                                                                  *
*  Callback so that the File -> New -> Library from the libManager   *
*  calls Virtuoso's File -> New -> Library. This means that the library  *
* will be created from within Virtuoso so the postCreateLib trigger *
*    will get invoked, for example. Most of the arguments here are    *
*                               ignored.                           *
*                                                                  *
*******************************************************************/

procedure( abCreateNewLib(menuName _l _c _v _f _cat)
  when(equal(menuName "abNewLib") ddsHiCreateLibrary())
  )
```

Example 5: Dynamically Sensitizing Menu Items in a Design Management Workarea

In this particular example, the capabilities of the function shown in Example 1 are extended so that certain menus can be sensitized within a design management workarea, based on whether the library currently selected is managed or not.

When either the pop-up menu is selected or the design manager pull-down menu is selected, the specified map callback is started in Virtuoso. This then sensitizes the menu items from examples 2 and 3 depending on whether the selected library is under DM control or not. It is vital for this kind of pre-map callback to be fast so that it can make the modification prior to the menu being displayed without blocking the window system.

1. Add the following code to the `cdsLibMgr.il` file:

```
    lmgrCreateMenu("popup" '(("mapCallback" ("abPopupCallback"))))
    lmgrCreateMenu("designCascade" '(("mapCallback" ("abPopupCallback"))))
```

2. Define the following function in Virtuoso:

```
/******************************************************************
*                                                                 *
*           abPopupCallback(_menuName lib _c _v _f _cat)          *
*                                                                 *
* pre-map callback to be started from either pop-up or design manager *
* pull-down. This sensitizes two of the menu items based on whether  *
*               this library is DM managed or not.                *
*                                                                 *
******************************************************************/
```

```
procedure( abPopupCallback(_menuName lib _c _v _f _cat)
lmgrSensitizeMenuItems('("ShowAllCheckOuts" "ShowUpdates")
    lib!="" && ddAmUsingDM(ddGetObj(lib)))
    t
    )
```

Example 6: Changing Labels and Fonts

You can use this sample function either in Virtuoso or in the `cdsLibMgr.il` file. In the following simple example, the label for *Copy Wizard* is changed to *Advanced Copy* and the font for *File – Open* is changed.

```
; change label for Copy Wizard
lmgrSetObject("editCopyButton" '(("label" "Advanced Copy...")))
;; choose silly font for File->Open
lmgrSetObject("FileOpenButton"
'(("font"
"-adobe-helvetica-medium-o-normal--12-120-75-75-p-67-iso8859-1"))
)
```

Example 7: Adding Toggles and Radio Button Fields

This example does not perform any useful task, but it demonstrates how you use the other menu types that are available. It creates a new pull-down menu in the menu bar, *GUI Objects*, which contains a toggle menu underneath, and then a submenu containing three radio options. The callback to Virtuoso causes a message to be displayed on the Library Manager output pane reflecting the current settings.

Note: The radio options must be in their own menu.

1. Add the following code to the `cdsLibMgr.il` file:

```
; create the toggle button
lmgrCreateMenuItem("ToggleExample" "toggle"
    '(("label" "Toggle Example")
      ("callback" ("abToggleExample"))
      )
    )

; create a new pull-down menu, to contain the toggle example
lmgrCreateMenu("GUIcascade" '(("label" "GUI Objects")))

; Add the toggle example into the new pull-down menu
lmgrAddMenuItems("GUIcascade" "" '("ToggleExample"))

; Add the new pull-down menu into the menu banner
lmgrAddMenuItems("menuBar" "" '("GUIcascade"))

; Create Three radio options
lmgrCreateMenuItem("RadioOption1" "radio"
    '(("label" "Option 1")
      ("callback" ("abRadioExample"))
      )
    )
lmgrCreateMenuItem("RadioOption2" "radio"
    '(("label" "Option 2")
      ("callback" ("abRadioExample"))
      )
    )
lmgrCreateMenuItem("RadioOption3" "radio"
    '(("label" "Option 3")
      ("callback" ("abRadioExample"))
      )
    )

; Create a pull-down to contain the three radio options
lmgrCreateMenu("RadioPulldown" '(("label" "Radio Pulldown")))
; Add the radio options to the pull-down. Note: The radio options must
; be in their own submenu
lmgrAddMenuItems("RadioPulldown" "" '("RadioOption1" "RadioOption2"
     "RadioOption3"))

lmgrAddMenuItems("GUIcascade" "" '("RadioPulldown"))
```

2. Use the following callbacks in Virtuoso:

```
/*****************************************************************
*                                                                *
*           abToggleExample(_menuName _l _c _v _f _cat)          *
*                                                                *
* Just prints a message saying what the current state of the toggle *
* setting is                             *
*                                                                *
*****************************************************************/

procedure( abToggleExample(_menuName _l _c _v _f _cat)
  lmgrDisplayMessage(
    sprintf(nil "Toggle set to %L\n"
      ; note - uses assoc to lookup state information
      cadr(assoc("state" lmgrGetObject("ToggleExample"))))
      )
  )

/*************************************************************
*                                                            *
*          abRadioExample(menuName _l _c _v _f _cat)         *
*                                                            *
*    Displays which radio option was selected or unselected  *
*                                                            *
*************************************************************/

procedure( abRadioExample(menuName _l _c _v _f _cat)
  lmgrDisplayMessage(
    sprintf(nil "Radio %s %sselected\n" menuName
      if( cadr(assoc("state" lmgrGetObject(menuName))) "" "un")
    ))
  )
```

5

Plotter Functions

This chapter provides syntax, descriptions, and examples for the Cadence SKILL functions associated with the configuation of plotters.

psQueueStatus

```
psQueueStatus( )
     => t
```

Description

Displays the plot jobs in the spooling queues.

Arguments

None.

Value Returned

Always returns `t`.

psConfigLoaded

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

Description

Indicates if the `.cdsplotinit` is already loaded.

Arguments

None.

Value Returned

`t`	Returns `t` if the `.cdsplotinit` is already loaded.
`nil`	Returns `nil` if the `.cdsplotinit` is not loaded.

psLoadCdsPlotInit

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

Description

Indicates if the `.cdsplotinit` has successfully loaded or not.

Arguments

None.

Value Returned

`t`	Returns `t` if the `.cdsplotinit` successfully loaded.
`nil`	Returns `nil` if the `.cdsplotinit` did not load.

psQueryPlotters

```
psQueryPlotters( )
     => l_plotters
```

Description

Returns a list of plotter names from the loaded `.cdsplotinit` files.

Arguments

None.

Value Returned

`l_plotters`	Returns a list of plotter names from all `.cdsplotinit` files that are loaded.

psQueryPaperSizes

```
psQueryPaperSizes(
     t_plotterName
     )
     => l_paperSizes
```

Description

Returns the paper sizes (such as A size) and offset of the paper that the plotter uses.

Arguments

t_plotterName	Name of the plotter.

Value Returned

l_paperSizes	Returns the paper sizes and offset of the paper that the plotter uses.

psQueryPaperSize

```
psQueryPaperSize(
     t_plotterName
     t_paperSize
     )
     => l_dimensions
```

Description

Returns the dimensions of the specified paper size.

Arguments

t_plotterName	Name of the plotter.
l_paperSizes	The paper size to query for dimensions.

Value Returned

l_dimensions	Returns the dimensions of the given paper size in inches.