Allegro X System Capture Tcl Commands

Product Version 23.1
September 2023

Contents

1

Tcl Commands in System Capture

System Capture includes a vast collection of Tcl commands that are used internally when designers perform tasks using its interface menu options. These Tcl commands are also available to anyone who needs to create scripts to modify the default built-in functionality or wants to accomplish design tasks without clicking and navigating the user interface.

ⓘ Tk Support
System Capture does not support Tk.

Scope of this Document

This document lists the Tcl commands that are available and describes how to use them. Knowledge of Tcl concepts is mandatory to be able to use and extend the System Capture Tcl commands. To learn the basics of Tcl commands, as in the language syntax, operands, assignments, substitutions, and so on, refer to https://www.tcl.tk/man/tcl8.6/TclCmd/info.htm. Many of the Tcl commands are for Internal use only and details for their behavior and parameters are not disclosed. Do not use these commands directly on your designs. If you require the functionality these commands offer, contact a Cadence representative for information and help. It is recommended you use only those commands that are listed in this document.

Prerequisites

Before you implement System Capture Tcl commands, you should be familiar with the following:

- Basics of schematic design tasks
- Fundamentals of System Capture
- Basics of Tcl scripting

Tk is not supported in System Capture. To create custom UIs, use HTML5/JavaScript. For more information on custom UIs, refer to application notes and samples available in the SPB installation.

Terminology

Term	Description
Component	Any of the basic parts used in building electronic equipment, such as a resistor, capacitor, DIP, or connector.
Symbol	The symbolic representation of a library component that you add to your design. This drawing defines the shape, pins, and general properties of the library component.
Package	A physical part that contains more than one logical part. For example, a 2N3905 transistor, a fuse, and a 74LS00 are packages.
Instance	A part or a symbol that you have placed on a schematic page.
Design	A schematic drawing created in System Capture.
Project	The work area for a design, including all the views of the design, links to libraries, and setup information such as Physical Part Table, configuration, and expansion directives. Separate directories exist for each design project.
DBID	The unique ID of a part in the System Capture design database. Means database id.
sPath	sPaths are unique string identifiers for block, page, instance, nets, and route object in a project.

Before You Begin

Before you start using the Tcl commands, here are some System Capture topics that you need to familiarize yourself with.

System Capture Object Types

A System Capture schematic has the following object types on which various operations are performed:

Schematic Canvas Object Types		**Connectivity and Electrical Object Types**	

Graphical	**Text**	**Electrical**	**Connectivity**
Basic • Line • Ellipse • Rectangle	• Properties • Navigation links • Signal names • Notes	Signals • Scalar • Bit scalar • Bus • NetGroup	Logical View Objects • Cell Interface Object • IDesign Object • IInstance Object • INet Object • ITerm Object • Net Bit Context • Instance Terminal Bit Context • Connection Object • Instance Terminal Context

Block • Shapes • Connectors		Blocks • Hierarchical blocks • System blocks	Physical View Objects • Physical part instance • Physical Net • Physical Pin Instance • Physical Function Instance • Physical Part Definition Iterator • Physical Part Instance Iterator • Physical Pin Iterator • Physical Net Iterator • Physical function Iterator
Other • Page border • Component bodies		Components • Parts from libraries	

		Special bodies • Power • Ground • Alias • Ports • Off page • NC • 1 to n • Comment	

Retrieving Design Object Information

There are three aspects of a System Capture design:

- Graphical
- Logical
- Physical

The purpose of logical view and physical view sections is to enable schematic designers and engineers working with System Capture to retrieve physical and logical information about the design through supported TCL APIs. That is, commands can be used to read the information about the design and the design objects, such as component instances and nets, loaded on the connectivity server. No modifications can be made to the design objects or their properties. The logical view and physical view sections of the document discuss commands related to logical and physical traversal respectively. The graphical aspect gives the graphical information associated with the schematic canvas. Modifications can be made to the design through commands supported in the graphical view.

Commonly Performed Tasks

Designers perform the following actions that modify the design components and control the display or aspects of the design.

Modifies the Design Components	Does not Modify Design Database

Adds objects
• *Electrical Objects*
  ◦ Wires, Buses, Net-groups
  ◦ Components
  ◦ Hierarchical Blocks
  ◦ System Design Objects
  ◦ Net symbols
• *Text Objects*
  ◦ Signal names
  ◦ Properties
  ◦ Rich Notes
  ◦ Custom Variables
• *Graphical Objects*
  ◦ Notes
  ◦ Basic Shapes
  ◦ Block Shapes
  ◦ Connectors
	Controls the display
	• Zoom
	• Pan
	• Drawing Guides
	• Selection

Modifies Objects • Move • Resize • Delete • Rotate • Mirror • Align • Distribute • Transparency	Navigates the Design • Ascend the hierarchy • Descend the hierarchy • Go to the next Page • Go to the previous Page • Navigate to a specific object
Launches Utilities • Ref-Des Management • Variant Management • Part Management • Find/Replace • Property Management	Launches Utilities • Print the design • Setup preferences • Create variants • Physical Net Name View

Common Data Types

The following data types are used for Tcl-related features, such as parameter type and the command's return type.

- INT
- STRING
- BOOL/BOOLEAN
- LIST
- DBID
- SPATH
- COLOR

- POINT
- BBOX
- CNSOBJID
- CNSOBJTYPE
- NONE

Using the Tcl Commands

In this section, standard Tcl commands are used to extract the details from the built-in Tcl tools and commands within System Capture. Only the relevant commands and features for System Capture are listed and explained. System Capture Tcl commands are based on Tcl version 8.6. There is no Tk support.

Running Tcl Commands

1. Open the Command Window by choosing *View* - *Command Window*
2. Type the command at the Tcl> prompt.

System Capture Name Spaces

To find out the namespaces available in System Capture, run the following command at the Tcl prompt in the Command Window: TcL>namespace children   The namespaces provided by System Capture are:

- cps - Workspace handling, common functionality
- sch - Schematic canvas commands
- cpb - Flow-related commands
- sdaUtils - Utility functions
- sdaUI - User interface-related functionality used for creating hybrid widgets
- cpCommon - Common functionality
- sdaConn - Connectivity-related commands

Commands Available in a Name Space

To list the commands available within a namespace, the Tcl info commands are used. For example:

Command	Displays
info commands sch::*	All commands in the sch namespace
info commands cps::*	All commands in the cps namespace
info commands sch::dbGet*	Lists all commands in the sch namespace starting with dbGet
info vars sch::DB*	Lists all variables in the sch namespace starting with DB

Mapping Design Elements with Tcl Types

Every object on the canvas has a type in Tcl. Here is an example to determine whether an object is a wire, the following snippet is used.

```
# Checks for a route object.
# Returns $::sch::DBTrue (value 1) if the
# dbId is of a route object,
# otherwise $::sch::DBFalse (value 0)
proc isWire {dbId} {
 set ret $::sch::DBFalse
 if {[::sch::dbGetType $dbId] == $::sch::DBTRoute} {
  set ret $::sch::DBTrue
 }
 return $ret
}
```

This table lists the mappings from object types to Tcl variables and values.

Object Type	Tcl Type	Numeric Value
Invalid Type	sch::DBTInvalid	-1
Point	sch::DBTPoint	0

Bounding box	sch::DBTBox	1
Note	sch::DBTNote	2
Rich note	sch::DBTRichNote	3
Simple Note	sch::DBTSimpleNote	4
Occurrence properties	sch::DBTOccurrenceProp	5
Line	sch::DBTLine	6
Ellipse	sch::DBTEllipse	7
Rectangle	sch::DBTRect	8
Arc	sch::DBTArc	9
Block	sch::DBTBlock	13
Connector	sch::DBTConnector	14
Page Border	sch::DBTGraphicInstance	15
Any instance type	sch::DBTInst	16
Power, Ground, Off-page, Port, Alias, No-connect	sch::DBTNetInstance	17
Part instance	sch::DBTInstance	18
Off-page	sch::DBTOffPage	19
Port	sch::DBTPort	20
Power	sch::DBTPower	21
Comment Body	sch::DBTCommentBody	22
Hierarchical block instance	sch::DBTFunctionBlock	23
Bus tap	sch::DBTBusTap	24
Alias	sch::DBTAlias	25
Route	sch::DBTRoute	26

Bus	sch::DBTBus	27
Wire segment	sch::DBTWireSegment	28
Connection Line	sch::DBTConnectLine	29
Bus segment	sch::DBTBusSegment	30
Junction on a wire	sch::DBTJunction	31
Component instance pin	sch::DBTInstTerm	32
Property	sch::DBTProp	35
Displayed property	sch::DBTDisplayProp	36
Net	sch::DBTNet	37
Page	sch::DBTPage	39
Table	sch::DBTTable	41
Image	sch::DBTPixMap	49
Routes in a By-pass item	sch::DBTDTRoute	50
Group	sch::DBTGroupItem	51
By-pass item	sch::DBTByPassItem	52
NetGroup	sch::DBTNetGroup	58
Block Pin	sch::DBTBlockPin	59
No Connect	sch::DBTNoConnect	60
List of DBIDs	sch::DBTList	43
List of properties	sch::DBTPropList	44
List of wire/bus segments	sch::DBTSegmentList	45
List of points	sch::DBTPointList	46
List of pairs of strings	sch::DBTStrMap	47

List of strings	sch::DBTStringList	48

The numeric values shown in the table might change across releases. Therefore, do not hardcode them in scripts. Use Tcl variables instead. See the following code sample:

```
set selItem [sch::dbGetSelectedItems [sch::dbGetActivePage]]
set selItemType [sch::dbGetType $selItem]
#incorrect usage and the number 26 is being hard-coded
if {26 == $selItemType} {
    puts "Selected object is a wire"
}

set selItem [sch::dbGetSelectedItems [sch::dbGetActivePage]]
set selItemType [sch::dbGetType $selItem]
# Recommended usage
if {$sch::DBTRoute == $selItemType} {
    puts "Selected object is a wire"
}
```

Boolean Mapping

Type	Tcl Type	Value
True value	DBTrue	1
False value	DBFalse	0

Typographic and Syntax Conventions

The conventions used in the documentation of Tcl commands are:

Use	For
|	Showing alternative parameters in the command syntax
[ ]	List items
{ }	Encloses strings that have special characters, such as whitespace in signal names
? ?	Optional parameter
< >	Mandatory parameter

Here are some examples.

```
cnsGetBusParentClass <bus_id> ?TraverseHier? ?design_name?
```

where,

- TraveseHier and design_name are optional parameters
- bus_id is mandatory

```
xnetPinPairdefinition -add ?-pins  <list_of_pin_pairs>? | -del <pin_pair> <XNetNum> | -
source <source_spath> -copyTo [list <dest_spath>] | -reset [list <obj_spath>]
```

where,

- At one time, the command can be used to add, del, source, or reset.
- -pins is an optional parameter for the add command
- Square brackets are used for a list, such as [list <dest_spath>]

Accessing Help

To access detailed information for the Tcl command, type the following in the Command Window:

```
help <command name>
```

For example,

```
Tcl> help addActionToContextMenu
```

The CadenceHelp window opens if the required document files are available and indexed.

2

Constraints Management

Constraints Management allows querying constraints and constraint objects associated with the electrical objects such as nets, buses, and NetGroups. The purpose of these TCL commands is to provide a constraint object list, parent and child objects associated with a specific constraint object. It also provides utility commands for:

- Validating the system objects, such as XNets and Differential Pairs
- Automatic creation of differential pairs
- Importing constraint data such as technology data, ECset data, pin delay information
- XNet specification on instances

asda_get_xnet

Returns the name of the XNet of a wire whose dbID is passed.

Return Type

STRING

Syntax

```
asda_get_xnet <dbId of route item>
```

Parameters

Parameter	Type	Description
`dbID`	`DBID`	Database identifier of a route This parameter is required.

Examples

```
#Commands to get the XNet name for the selected route on the current page
#set item [ sch::dbGetSelectedItems [ sch::dbGetActivePage ]]
#puts $item
asda_get_xnet $item
#Returns db:00000036
#Returns SATA_TX_P&lt;4&gt;
```

Related Commands

dbGetSelectedItems

dbGetActivePageSPath

cnsAutoCreateDiffPair

It creates differential pairs based on the signal names patterns specified in the configuration file.

Return Type

BOOLEAN

Syntax

```
cnsAutoCreateDiffPair
```

Examples

```
cnsAutoCreateDiffPair
```

cnsFindObjConstraint

Use cnsFindObjConstraint to find the constraint properties on the specified object. It returns a list containing the constraint property values, the isInherited flag value (1 or 0), and the units of the property value.

Return Type

LIST

Syntax

```
cnsFindObjConstraint <design_name> <object_spath> <object_type> <prop_name>
```

Parameters

Parameter	Type	Description
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). This parameter is required.
`object_spath`	`SPATH`	The spath of the object in a design. The value must be within straight double quotes (" "). This parameter is required.
`obj_type`	`CNSOBJTYPE`	The type of object. The value must be in uppercase and enclosed within straight double quotes (" "). This parameter is required.
`prop_name`	`STRING`	The property name. The value must be in uppercase and enclosed within straight double quotes (" "). This parameter is required.

Examples

```
#command to find the DIFFP_GATHER_CONTROL property on @worklib.aaa(tbl_1):\\O1\\
#net in documentation design.
cnsFindObjConstraint documentation @worklib.aaa(tbl_1):\\O1\\ NET DIFFP_GATHER_CONTROL
```

Related Commands

dbGetSPath

cnsGetBusConstraint

It returns a list of parent design objects for the bus, on which specified constraint property exists.

Return Type

LIST

Syntax

```
cnsGetBusConstraint <bus_id> <prop_name> ?design_name?
```

Parameters

Parameter	Type	Description
`bus_id`	`CNSOBJID`	The ID of specified BUS object. This parameter is required.
`prop_name`	`STRING`	The property name. The value must be in uppercase and enclosed within straight double quotes (" "). This parameter is required.
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). When no value is specified, root design is used as the design name. This parameter is optional.

Examples

```
#command to find the "PHASE_TOL" property on dbc:0x00000004 in the Root design
cnsGetBusConstraint dbc:0x00000004 PHASE_TOL
```

Related Commands

dbGetSelectedItems

dbGetRootDesignName

cnsGetBusConstraintSet

cnsGetBusConstraintSet

It returns a list of the constraint sets applied on the specified bus object in a design.

Return Type

LIST

Syntax

```
cnsGetBusConstraintSet <bus_id> <cns_type> ?design_name?
```

Parameters

Parameter	Type	Description
`bus_id`	`CNSOBJID`	The ID of specified bus object. This parameter is required.
`cns_type`	`CNSOBJTYPE`	The constraint set type (such as ECSET, PCSET, SCSET) This parameter is required.
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). If this value is not specified, by default, the root design name is used. This parameter is optional.

Examples

```
#Find the Electrical Constraint Set Object applied on the specified bus object in the
root design.
cnsGetBusConstraintSet dbc:0x00000004 "ECSET"
```

Related Commands

dbGetSelectedItems

dbGetRootDesignName

cnsGetBusConstraint

cnsGetBusInDesign

It returns a list of buses present in the specified design.

Return Type

LIST

Syntax

```
cnsGetBusInDesign <design_name>
```

Parameters

Parameter	Type	Description
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). This parameter is required.

Examples

```
#command to retrieve the list of bus objects in the 'root_des' design
cnsGetBusInDesign "root_des"
```

Related Commands

dbGetRootDesignName

cnsGetBusConstraint

cnsGetBusConstraintSet

cnsGetBusInDesign

cnsGetBusMemberNet

cnsGetBusMemberXNet

cnsGetBusParentClass

cnsGetClassMemberBus

cnsGetBusMemberNet

It returns a list of member nets of the specified bus in the design.

Return Type

LIST

Syntax

```
cnsGetBusMemberNet <busID> ?design_name?
```

Parameters

Parameter	Type	Description
`busID`	`CNSOBJID`	The ID of specified bus object. This parameter is required.
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). If this value is not specified, by default, the root design name is used. This parameter is optional.

Examples

```
#command to list member nets of a bus with ID 'dbc:0x00000004'
cnsGetBusMemberNet dbc:0x00000004
```

Related Commands

dbGetSelectedItems

dbGetRootDesignName

cnsGetBusInDesign

cnsGetBusParentClass

Use cnsGetBusParentClass to get the parent class of a bus object. The valid values for bTraverseHier are 0 and 1. If bTraverseHier is 1 then if direct parent net class is not found for the bus, all parents of the bus are checked for a possible parent net class.

Return Type

LIST

Syntax

```
cnsGetBusParentClass <bus_id> ?bTraverseHier? ?design_name?
```

Parameters

Parameter	Type	Description
`bus_id`	`CNSOBJID`	The ID of the bus object. This parameter is required.
`bTraverseHier`	`INT`	The parameter to set traversal direction. If this parameter is not specified, default value of 1 is used. This parameter is optional. Default value is `1`.
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). If this parameter is not specified, by default, root design is used as the design name. This parameter is optional.

Examples

```
#command to get the Parent Net class of dbc:0x00000004 in the root design
cnsGetBusParentClass dbc:0x00000004
```

Related Commands

dbGetSelectedItems

dbGetRootDesignName

cnsGetBusConstraint

cnsGetBusConstraintSet

cnsGetBusInDesign

cnsGetBusMemberNet

cnsGetBusMemberXNet

cnsGetBusParentClass

cnsGetClassMemberBus

cnsGetClassConstraint

It returns a list of constraints applied to the specified net class.

Return Type

LIST

Syntax

```
cnsGetClassConstraint <classID> <prop_name> ?design_name?
```

Parameters

Parameter	Type	Description
`classID`	`CNSOBJID`	The ID of the specified net class This parameter is required.
`prop_name`	`STRING`	The property name. The value must be in uppercase and enclosed within straight double quotes (" "). This parameter is required.
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). If not given root design name is used as the default. This parameter is optional.

Examples

```
#command to find the "PHASE_TOL" property on dbc:00000013 in default root design
cnsGetClassConstraint dbc:00000013 PHASE_TOL
```

Related Commands

dbGetSelectedItems

dbGetRootDesignName

cnsGetClassConstraintSet

cnsGetClassConstraintSet

It returns a list of the constraint set applied to the specified class object.

Return Type

STRING

Syntax

```
cnsGetClassConstraintSet <classID> <cns_type> ?design_name?
```

Parameters

Parameter	Type	Description
`class_id`	`CNSOBJID`	The ID of the specified net class. This parameter is required.
`cns_type`	`CNSOBJTYPE`	The constraint set type. Valid values are ECSET, PCSET, and SCSET. This parameter is required.
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). When this parameter is not specified, root design is used as the design name. This parameter is optional.

Examples

```
#command to find the Electrical Constraint Set Object applied to the class object
specified
#by the ID, in the root design
cnsGetClassConstraintSet dbc:00000013 ECSET
```

Related Commands

dbGetSelectedItems

dbGetRootDesignName

cnsGetClassConstraint

cnsGetClassMemberBus

It returns a list of the bus members in the specified net class object in a design.

Return Type

LIST

Syntax

```
cnsGetClassMemberBus <class_id> ?design_name?
```

Parameters

Parameter	Type	Description
`class_id`	`CNSOBJID`	The ID of the net class This parameter is required.
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). If this parameter is not specified, root design is used as the design name. This parameter is optional.

Examples

```
#command to list the bus members of class object of the default root design
cnsGetClassMemberBus dbc:00000015
```

Related Commands

dbGetSelectedItems

dbGetRootDesignName

cnsGetClassConstraintSet

cnsGetClassMemberDiffPair

cnsGetClassMemberNet

cnsGetClassMemberXNet

cnsGetClassMemberDiffPair

It returns a list of differential pair members for the specified net class in the design.

Return Type

LIST

Syntax

```
cnsGetClassMemberDiffPair <classID> ?design_name?
```

Parameters

Parameter	Type	Description
`classID`	`CNSOBJID`	The unique identifier (ID) of the net class. This parameter is required.
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). If this parameter is not specified, root design is used as the design name. This parameter is optional.

Examples

```
#command to get the differential pair members of a net class in the root design
cnsGetClassMemberDiffPair dbc:00000015
```

Related Commands

dbGetSelectedItems

dbGetRootDesignName

cnsGetClassConstraintSet

cnsGetClassMemberBus

cnsGetClassMemberNet

cnsGetClassMemberXNet

cnsGetClassMemberNet

It returns a list of net members of the specified net class in the design.

Return Type

LIST

Syntax

```
cnsGetClassMemberNet <class_id> ?design_name?
```

Parameters

Parameter	Type	Description
`class_id`	`CNSOBJID`	The unique identifier (ID) of the net class. This parameter is required.
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). If this parameter is not specified, root design is used as the design name. This parameter is optional.

Examples

```
#command to get net members of a specific net class in the root design
cnsGetClassMemberNet dbc:00000015
```

Related Commands

dbGetSelectedItems

dbGetRootDesignName

cnsGetClassConstraintSet

cnsGetClassMemberBus

cnsGetClassMemberDiffPair

cnsGetClassMemberNet

cnsGetClassMemberXNet

It returns a list of xnet members in the specified net class in the design.

Return Type

LIST

Syntax

```
cnsGetClassMemberXNet <class_id> ?design_name?
```

Parameters

Parameter	Type	Description
`class_id`	`STRING`	The unique identifier (ID) of the net class. This parameter is required.
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). If this parameter is not specified, root design is used as the design name. This parameter is optional.

Examples

```
#command to get a list containing the XNet members of a specific
#net class in the root design.
cnsGetClassMemberXNet db:00000015
```

Related Commands

dbGetSelectedItems

dbGetRootDesignName

cnsGetClassConstraintSet

cnsGetClassMemberBus

cnsGetClassMemberDiffPair

cnsGetClassMemberNet

cnsGetDiffPairConstraint

Use cnsGetDiffPairConstraint to get the constraint set on a differential pair. It returns a list containing the constraint set applied.

Return Type

STRING

Syntax

```
cnsGetDiffPairConstraint <dpID> <cons_name> ?design_name?
```

Parameters

Parameter	Type	Description
`dpID`	`CNSOBJID`	The ID of the differential pair object. This parameter is required.
`cons_name`	`STRING`	The name of the constraints. The value must be in uppercase and enclosed within straight double quotes (" "). This parameter is required.
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). If not specified, then root design name is used as the default. This parameter is optional.

Examples

```
#command to find the constraint applied on a differential pair object
#in the default root design
cnsGetDiffPairConstraint dbc:0x00000004 DIFFP_PHASE_TOL
```

Related Commands

dbGetSelectedItems

dbGetRootDesignName

cnsGetDiffPairConstraintSet

cnsGetDiffPairConstraintSet

Use cnsGetDiffPairConstraintSet to get the constraint set on a net which is part of a differential pair. It returns a list of constraint sets of the specified type applied on differential pair member nets.

Return Type

LIST

Syntax

```
cnsGetDiffPairConstraintSet <dp_id> <cns_type> ?design_name?
```

Parameters

Parameter	Type	Description
`dp_id`	`CNSOBJID`	The ID of specified Differential Pair object. This parameter is required.
`cns_type`	`CNSOBJTYPE`	The constraint set type. Valid values are ECSET, PCSET, and SCSET. This parameter is required.
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). If this parameter is not specified, root design is used as the design name. This parameter is optional.

Examples

```
#command to find the Electrical Constraint Set Object
#applied on a specified differential pair object
#in the default root design
cnsGetDiffPairConstraintSet dbc:0x00000004 ECSET
```

Related Commands

dbGetSelectedItems

dbGetRootDesignName

cnsGetDiffPairConstraint

cnsGetDiffPairInDesign

Use cnsGetDiffPairInDesign to get all the differential pairs present in the design.

Return Type

STRING

Syntax

```
cnsGetDiffPairInDesign <design_name>
```

Parameters

Parameter	Type	Description
`design_name`	`STRING`	Name of the design This parameter is required.

Examples

```
#command to get the differential pairs
#on design named 'root_des'
cnsGetDiffPairInDesign root_des
```

cnsGetDiffPairMemberNet

Use cnsGetDiffPairMemberNet to get the list of the net members of the differential pair specified by ID in the design.

Return Type

STRING

Syntax

```
cnsGetDiffPairMemberNet <ID> ?design_name?
```

Parameters

Parameter	Type	Description
`ID`	`DBID`	DB ID of the differential pair This parameter is required.
`design_name`	`STRING`	Name of the design. If not specified then root design name is used as the default. This parameter is optional.

Examples

```
#command to get the members of a differential pair with ID db:00000015
cnsGetDiffPairMemberNet db:00000015
```

cnsGetDiffPairMemberXNet

Use cnsGetDiffPairMemberXNet to get the list of the xnet members of the differential pair specified by ID in the design.

Return Type

STRING

Syntax

```
cnsGetDiffPairMemberXNet <ID> ?design_name?
```

Parameters

Parameter	Type	Description
`ID`	`DBID`	DB ID of the differential pair. This parameter is required.
`design_name`	`STRING`	Name of the design. If not specified then root design name is used as the default. This parameter is optional.

Examples

```
#command to get the XNet members of a differential pair
#with ID db:00000015
cnsGetDiffPairMemberXNet db:00000015
```

cnsGetDiffPairParentClass

Use cnsGetDiffPairParentClass to get the list of net class of the parent objects of a differential pair specified by ID.
traversal_direction can be 0 or 1. A value of 0 means do not traverse the parents of the object if a direct parent net class is not found.

Return Type

LIST

Syntax

```
cnsGetDiffPairParentClass <ID> ?traversal_direction? ?design_name?
```

Parameters

Parameter	Type	Description
`ID`	`CNSOBJID`	The ID of a differential pair object This parameter is required.
`traversal_direction`	`INT`	The traversal direction This parameter is optional. Default value is `1`.
`design_name`	`STRING`	Name of the design. If not specified the root design name is used as default. This parameter is optional.

Examples

```
#command to get the list of parent net class of the differential pair specified by the
id db:00000013
cnsGetDiffPairParentClass db:00000013
```

cnsGetECSetInDesign

Use cnsGetECSetInDesign to get all the ECSets present in the design.

Return Type

STRING

Syntax

```
cnsGetECSetInDesign ?design_name?
```

Parameters

Parameter	Type	Description
`design_name`	`STRING`	Name of the design for which ECSets are being queried. If not specified then ECSets in the root design are returned. This parameter is optional.

Examples

```
#To get all ECSets in the root design
cnsGetECSetInDesign

#To get all ECSets in a design named memory
cnsGetECSetInDesign memory
```

cnsGetKey

Use cnsGetKey to get the sPath of the object with the specified ID.

Return Type

SPATH

Syntax

```
cnsGetKey <obj_id>
```

Parameters

Parameter	Type	Description
`obj_id`	`CNSOBJID`	Object ID This parameter is required.

Examples

```
#command to get the spath of an object
#with ID dbc:0x00000004
cnsGetKey dbc:0x00000004
```

cnsGetMatchGroupConstraint

Use cnsGetMatchGroupConstraint to get the constraint value for a specified constraint on a matched group. It returns a list containing the constraint values.

Return Type

LIST

Syntax

```
cnsGetMatchGroupConstraint <mg_id> <cons_name> ?design_name?
```

Parameters

Parameter	Type	Description
`mg_id`	`CNSOBJID`	The ID of the match group object This parameter is required.
`cons_name`	`STRING`	Name of the constraint This parameter is required.
`design_name`	`STRING`	Name of the design. If not specified then root design name is used as the default. This parameter is optional.

Examples

```
#command to get list of constraint values of specified match group in the root design
GetMatchGroupConstraint db:00000013 DIFFP_PHASE_TOL
```

cnsGetMatchGroupConstraintSet

Use cnsGetMatchGroupConstraintSet to get the constraint set on a net which is part of a match group. It returns a list containing the applied constraint set.

Return Type

LIST

Syntax

```
cnsGetMatchGroupConstraintSet <mg_id> <cs_type> ?design_name?
```

Parameters

Parameter	Type	Description
`mg_id`	`STRING`	The ID of the match group object. This parameter is required.
`cs_type`	`STRING`	The constraint set type ("ECSET", "PCSET", "SCSET") This parameter is required.
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). If not specified then root design name is used as the default. This parameter is optional.

Examples

```
#command to get the constraint set on nets in the match group
#with ID dbc:00000013
cnsGetMatchGroupConstraintSet dbc:00000013 ECSET
```

cnsGetMatchGroupInDesign

Use cnsGetMatchGroupInDesign to get all the match groups present in the design. It returns CNSOBJIDs for the match groups.

Return Type

CNSOBJID

Syntax

```
cnsGetMatchGroupInDesign <design_name>
```

Parameters

Parameter	Type	Description
`design_name`	`STRING`	Name of the design for which match groups are to be queried This parameter is required.

Examples

```
#command to get all MatchGroups defined in the design root_des
cnsGetMatchGroupInDesign root_des
```

Related Commands

cnsGetMatchGroupConstraint

cnsGetMatchGroupConstraintSet

cnsGetMatchGroupMemberNet

cnsGetMatchGroupMemberPinPair

cnsGetMatchGroupMemberXNet

cnsGetNetParentMatchGroup

cnsGetXNetParentMatchGroup

cnsGetMatchGroupMemberNet

It returns a list containing CNSOBJIDs of net members of a specific match group in the design.

Return Type

CNSOBJID

Syntax

```
cnsGetMatchGroupMemberNet <mgID> ?design_name?
```

Parameters

Parameter	Type	Description
`mgID`	`CNSOBJID`	The match group identifier This parameter is required.
`design_name`	`STRING`	The name of the design. If not specified root design name is used as the default. This parameter is optional.

Examples

```
#command to get the member nets of match group whose CNSOBJID is db:00000015
cnsGetMatchGroupMemberNet db:00000015
```

Related Commands

cnsGetMatchGroupConstraint

cnsGetMatchGroupConstraintSet

cnsGetMatchGroupInDesign

cnsGetMatchGroupMemberPinPair

It returns a list containing CNSOBJIDs of pin pair members of a specific match group in the design.

Return Type

CNSOBJID

Syntax

```
cnsGetMatchGroupMemberPinPair <mgID> ?design_name?
```

Parameters

Parameter	Type	Description
`mgID`	`CNSOBJID`	The match group identifier This parameter is required.
`design_name`	`STRING`	The name of the design. If not specified then root design name is used. This parameter is optional.

Examples

```
#command to get pin pairs of match group db:00000015 in root design
cnsGetMatchGroupMemberPinPair db:00000015
#command to get pin pairs of match group db:00000020 in design memory
cnsGetMatchGroupMemberPinPair db:00000020 memory
```

Related Commands

cnsGetMatchGroupConstraint

cnsGetMatchGroupConstraintSet

cnsGetMatchGroupInDesign

cnsGetMatchGroupMemberNet

cnsGetMatchGroupMemberXNet

cnsGetNetParentMatchGroup

cnsGetXNetParentMatchGroup

cnsGetMatchGroupMemberXNet

It returns a list containing CNSOBJIDs of XNet members of a specific match group in the design.

Return Type

CNSOBJID

Syntax

```
cnsGetMatchGroupMemberXNet <mgID> ?design_name?
```

Parameters

Parameter	Type	Description
`mgID`	`CNSOBJID`	The match group identifier This parameter is required.
`design_name`	`STRING`	The name of the design. If not specified then root design name is used. This parameter is optional.

Examples

```
#command to get the XNet members of the match group db:00000015 in the root design
cnsGetMatchGroupMemberXNet db:00000015
#command to get the XNet members of the match group db:00000025 in design memory
cnsGetMatchGroupMemberXNet db:00000025 memory
```

Related Commands

cnsGetMatchGroupConstraint

cnsGetMatchGroupConstraintSet

cnsGetMatchGroupInDesign

cnsGetMatchGroupMemberNet

cnsGetMatchGroupMemberPinPair

cnsGetNetParentMatchGroup

cnsGetXNetParentMatchGroup

cnsGetMemberXNet

Returns a list containing the XNet members of a specific constraint object in the design.

Return Type

LIST

Syntax

```
cnsGetMemberXNet <parentID> ?design_name?
```

Parameters

Parameter	Type	Description
`parentID`	`CNSOBJID`	Parent Constraint Object This parameter is required.
`design_name`	`STRING`	The name of the design. If not specified then root design name is used as the default. This parameter is optional.

Examples

```
#get XNet members of constraint object db:00000015 in the root design
#here constraint object can be a bus, match group, net class, net group
cnsGetMemberXNet db:00000015
```

cnsGetNetClassInDesign

Retrieves all the net classes present in the design in a list.

Return Type

LIST

Syntax

```
cnsGetNetClassInDesign <design_name>
```

Parameters

Parameter	Type	Description
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). This parameter is required.

Examples

```
#command to retrieve the list of bus objects in the 'root_des' design
cnsGetNetClassInDesign "root_des"
```

Related Commands

cnsGetClassConstraint

cnsGetClassConstraintSet

cnsGetClassMemberBus

cnsGetClassMemberDiffPair

cnsGetClassMemberNet

cnsGetClassMemberXNet

cnsGetNetConstraint

It returns a list of parent design objects for the net, on which specified constraint property exists.

Return Type

STRING

Syntax

```
cnsGetNetConstraint <netID> <cons_name> ?design_name?
```

Parameters

Parameter	Type	Description
`netID`	`CNSOBJID`	The ID of specified NET object. This parameter is required.
`cons_name`	`STRING`	The property name. The value must be in uppercase and enclosed within straight double quotes (" "). This parameter is required.
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). When no value is specified, root design is used as the design name. This parameter is optional.

Examples

```
#command to find the "PHASE_TOL" property on a net in the Root design
#set listNets [ cnsGetNetInDesign "root_des"]
#set net [lindex $listNets 0 ]
cnsGetNetConstraint $net "MIN_LINE_WIDTH" "dak_1"
#Return Value
#{77.00 mil 1}
```

Related Commands

dbGetSelectedItems

cnsGetNetConstraintSet

It returns a list of the constraint sets applied on the specified net object in the specified domain (Electrical/Physical/Spacing).

Return Type

LIST

Syntax

```
cnsGetNetConstraintSet <netID> <cs_type> ?design_name?
```

Parameters

Parameter	Type	Description
`netID`	`CNSOBJID`	The ID of specified NET object. This parameter is required.
`cs_type`	`STRING`	The constraint set type (such as "ECSET", "PCSET", "SCSET") This parameter is required.
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). If this value is not specified, by default, the root design name is used. This parameter is optional.

Examples

```
#command to find the Electrical Constraint Set Object
#applied on specified Netobject object in the root design
cnsGetNetConstraintSet dbc:0x00000004 "ECSET"
```

Related Commands

dbGetSelectedItems

dbGetRootDesignName

cnsGetNetConstraint

cnsGetNetInDesign

It returns a list of all the nets present in the specified design.

Return Type

LIST

Syntax

```
cnsGetNetInDesign <design_name>
```

Parameters

Parameter	Type	Description
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). This parameter is required.

Examples

```
#command to retrieve the list of bus objects in the 'root_des' design
cnsGetNetInDesign "root_des"
```

Related Commands

cnsGetNetConstraint

cnsGetNetConstraintSet

cnsGetNetParentBus

cnsGetNetParentClass

cnsGetNetParentDiffPair

cnsGetNetParentMatchGroup

cnsGetNetParentXNet

cnsGetNetMemberPinPair

cnsGetNetMemberPinPair

It returns a list of pin pair member objects for the specified net in the design.

Return Type

STRING

Syntax

```
cnsGetNetMemberPinPair <netID> ?design_name?
```

Parameters

Parameter	Type	Description
`netID`	`CNSOBJID`	The ID of specified NET object This parameter is required.
`design_name`	`STRING`	The name of the design. If not specified, then root design name is used. This parameter is optional.

Examples

```
#command to get pin pair members of the net db:00000005 in the root design
cnsGetNetMemberPinPair db:00000005
#command to get pin pair members of the net db:00000015 in the design called memory
cnsGetNetMemberPinPair db:00000015 memory
```

Related Commands

cnsGetPinPairsInDesign

cnsGetMatchGroupMemberPinPair

cnsGetNetParentBus

It returns a list containing the CNSOBJID of the parent of a specific net of bus type.

Return Type

CNSOBJID

Syntax

```
cnsGetNetParentBus <netID> ?bTraverseHier? ?design_name?
```

Parameters

Parameter	Type	Description
`netID`	`CNSOBJID`	The net identifier. This parameter is required.
`bTraverseHier`	`INTEGER`	The parameter to set the traversal direction. This parameter is optional. Default value is `1`.
`design_name`	`STRING`	The name of the design. If not specified, then root design name is used. This parameter is optional.

Examples

```
#comamnd to get the parent bus for the given net id of a net of the root design
cnsGetNetParentBus db:00000013
#command to get the parent bus for the given net id of a net of the design memory
cnsGetNetParentBus db:00000030 memory
```

Related Commands

cnsGetClassMemberNet

cnsGetDiffPairMemberXNet

cnsGetMatchGroupMemberNet

cnsGetNetParentClass

It returns a list containing the CNSOBJIDs of parents of a specific net of the net class type.

Return Type

CNSOBJID

Syntax

```
cnsGetNetParentClass <netID> ?bTraverseHier? ?design_name?
```

Parameters

Parameter	Type	Description
`netID`	`CNSOBJID`	The net identifier. This parameter is required.
`bTraverseHier`	`INT`	The parameter to set the traversal direction. This parameter is optional. Default value is `1`.
`design_name`	`STRING`	The name of the design. If not specified, then root design name is used. This parameter is optional.

Examples

```
#command to get the parent net class for the given net id of a net in the root design
cnsGetNetParentClass db:00000013
#command to get the parent net class for the given net id of a net in the design memory
cnsGetNetParentClass db:00000040 memory
```

cnsGetNetParentDiffPair

It returns the CNSOBJID of the parent differential pair of the specified net.

Return Type

CNSOBJID

Syntax

```
cnsGetNetParentDiffPair <netID> ?bTraverseHier? ?design_name?
```

Parameters

Parameter	Type	Description
`netID`	`CNSOBJID`	The net identifier. This parameter is required.
`bTraverseHier`	`INT`	The parameter to set the traversal direction. This parameter is optional. Default value is `1`.
`design_name`	`STRING`	Name of the design. If not specified, root design is used. This parameter is optional.

Examples

```
#command to get parent differential pair of the net with the given net id in the root
design
cnsGetNetParentDiffPair db:00000013
#command to get parent differential pair of the net with the given net id in the design
memory
cnsGetNetParentDiffPair db:00000031 memory
```

cnsGetNetParentMatchGroup

It returns a list containing the CNSOBJIDs of the parent match groups of a specific net.

Return Type

CNSOBJID

Syntax

```
cnsGetNetParentMatchGroup <netID> ?bTraverseHier? ?design_name?
```

Parameters

Parameter	Type	Description
`netID`	`CNSOBJID`	The net identifier. This parameter is required.
`bTraverseHier`	`INT`	The parameter to set traversal direction. This parameter is optional. Default value is `1`.
`design_name`	`STRING`	The name of the design. If not specified then root design name is used. This parameter is optional.

Examples

```
#command to get parent match group for the given net in the root design
cnsGetNetParentMatchGroup db:00000013
#command to get parent match group for the given net in the design memory
cnsGetNetParentMatchGroup db:00000031 memory
```

Related Commands

cnsGetMatchGroupConstraint

cnsGetMatchGroupConstraintSet

cnsGetMatchGroupInDesign

cnsGetMatchGroupMemberNet

cnsGetMatchGroupMemberPinPair

cnsGetMatchGroupMemberXNet

cnsGetXNetParentMatchGroup

cnsGetNetParentXNet

It returns a list of parent XNets of the specified net in the design.

Return Type

STRING

Syntax

```
cnsGetNetParentXNet <netID> ?bTraverseHier? ?design_name?
```

Parameters

Parameter	Type	Description
`netID`	`CNSOBJID`	The ID of specified net object. This parameter is required.
`bTraverseHier`	`INTEGER`	The parameter to set the traversal direction. This parameter is optional. Default value is `1`.
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). If this value is not specified, by default, the root design name is used. This parameter is optional.

Examples

```
#command to list of Parents XNet of a Net with ID 'dbc:0x00000004' in the root design
cnsGetNetParentXNet dbc:0x00000004
```

Related Commands

dbGetSelectedItems

dbGetRootDesignName

cnsGetBusInDesign

cnsGetObjConstraintSet

Use cnsGetObjConstraintSet to find the constraint properties on the specified object. It returns a list containing the constraint property values, the isInherited flag value (1 or 0), and units of the property value.

Return Type

LIST

Syntax

```
cnsGetObjConstraintSet <design_name> <object_kind> <object_spath> <constraint_set_kind>
```

Parameters

Parameter	Type	Description
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). This parameter is required.
`object_kind`	`CNSOBJTYPE`	The type of object. The value must be in uppercase and enclosed within straight double quotes (" "). This parameter is required.
`object_spath`	`STRING`	The spath of the object in a design. The value must be within straight double quotes (" "). This parameter is required.
`constraint_set_type`	`STRING`	The type of the constraint set. This value must be in uppercase and enclosed within straight double quotes (" "). This parameter is required.

Examples

```
#command to find the "DIFFP_GATHER_CONTROL" property on
#"@worklib.aaa(tbl_1):\\O1\\" Net in "documentation" design
cnsFindObjConstraint "documentation" "NET" "@worklib.aaa(tbl_1):\\O1\\"
"DIFFP_GATHER_CONTROL"
```

Related Commands

dbGetSPath

cnsGetObjInDesign

Use cnsGetObjInDesign to get the specified object in the design. It returns a list containing the specified object in the design.

Return Type

LIST

Syntax

```
cnsGetObjInDesign <design_name> <obj_kind>
```

Parameters

Parameter	Type	Description
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). This parameter is required.
`obj_kind`	`CNSOBJTYPE`	The type of object. The value must be in uppercase and enclosed within straight double quotes (" "). This parameter is required.

Examples

```
#command to find "NET" type of object in design "abc"
cnsGetObjInDesign "abc" "NET"
```

Related Commands

dbGetSPath

cnsGetObjMember

Use cnsGetObjMember to get the member kind of the specified object. This command will return the list of the members of the specified object in the design.

Return Type

LIST

Syntax

```
cnsGetObjMember <design_name> <object_type> <path_of_object> <member_type>
```

Parameters

Parameter	Type	Description
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). This parameter is required.
`object_type`	`CNSOBJTYPE`	The type of object. The value must be in uppercase and enclosed within straight double quotes (" "). This parameter is required.
`path_of_object`	`SPATH`	The spath of the object in a design. The value must be within straight double quotes (" "). This parameter is required.
`member_type`	`CNSOBJTYPE`	The Member type of object. The value must be in uppercase and enclosed within straight double quotes (" "). This parameter is required.

Examples

```
#command to find the "NET" member of specified "DIFFPAIR"
#"@worklib.aaa(tbl_1):\\O1\\" in design "abc"
cnsGetObjMember "abc" "DIFFPAIR" "@worklib.aaa(tbl_1):\\O1\\" "NET"
```

Related Commands

dbGetSPath

cnsGetObjParent

Use this command of find Kind of parents in specified kind of object in design, it will return the list of kind Parents of specified object.

Return Type

LIST

Syntax

```
cnsGetObjParent <design_name> <type_of_object> <path_of_object> <bTraverseHier>
<parent_kind>
```

Parameters

Parameter	Type	Description
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). This parameter is required.
`type_of_object`	`CNSOBJTYPE`	The type of the object.This value must be enclosed within straight double quotes (" "). This parameter is required.
`path_of_object`	`SPATH`	The spath of the object in a design. The value must be within straight double quotes (" "). This parameter is required.
`bTraverseHier`	`INT`	Direction of the traversal. This value must be enclosed within straight double quotes (" "). This parameter is required.
`parent_kind`	`CNSOBJTYPE`	The object type of the parent. The value must be specified in uppercase and enclosed within straight double quotes (" "). This parameter is required.

Examples

```
#command to get "DIFFPAIR" parent of specified "NET"
#"@worklib.aaa(tbl_1):\\N25\\(5)" in design "abc"
cnsGetObjParent "abc" "NET" "@worklib.aaa(tbl_1):\\N25\\(5)" "1" "DIFFPAIR"
```

Related Commands

dbGetSPath

cnsGetPCSetInDesign

Returns a list of CNSOBJIDs of all the Physical Constraint Sets in the design.

Return Type

CNSOBJID

Syntax

```
cnsGetPCSetInDesign <design_name>
```

Parameters

Parameter	Type	Description
`design_name`	`STRING`	Name of the design for which physical constraint sets are to be queried This parameter is required.

Examples

```
#command to get all the physical constraint sets in design root_des
cnsGetPCSetInDesign root_des
```

cnsGetPinPairsInDesign

Gets all the pin pairs present in the design.

Return Type

STRING

Syntax

```
cnsGetPinPairsInDesign <design_name>
```

Parameters

Parameter	Type	Description
`design_name`	`STRING`	Name of the design for which pin pairs are to be queried. This parameter is required.

Examples

```
#get all pin pairs in the design sda_root
cnsGetPinPairsInDesign sda_root
```

cnsGetSCSetInDesign

Returns CNSOBJIDs of all the spacing constraint sets in the design.

Return Type

CNSOBJID

Syntax

```
cnsGetSCSetInDesign <design_name>
```

Parameters

Parameter	Type	Description
`design_name`	`STRING`	Name of the design This parameter is required.

Examples

```
#command to get all the spacing constraint sets in the design root_des
cnsGetSCSetInDesign root_des
```

cnsGetXNetConstraint

Returns the value of a constraint on the specified XNet.

Return Type

STRING

Syntax

```
cnsGetXNetConstraint <xnetID> <cons_name> ?design_name?
```

Parameters

Parameter	Type	Description
`xnetID`	`CNSOBJID`	The XNet identifier This parameter is required.
`cons_name`	`STRING`	The constraint name This parameter is required.
`design_name`	`STRING`	The name of the design. If not specified then root design name is used. This parameter is optional.

Examples

```
#command to find STUB_LENGTH on a given XNet
cnsGetXNetConstraint db:00000013 STUB_LENGTH
```

Related Commands

cnsGetXNetInDesign

cnsGetXNetConstraintSet

Use cnsGetXNetConstraintSet to return the name of the constraint set of the specified type (Electrical, Physical, or Spacing) on the XNet.

Return Type

STRING

Syntax

```
cnsGetXNetConstraintSet <xnetID> <cs_type> ?design_name?
```

Parameters

Parameter	Type	Description
`xnetID`	`DBID`	The XNet identifier This parameter is required.
`cs_type`	`STRING`	The constraint set type. Valid values are ECSET, PCSET, SCSET This parameter is required.
`design_name`	`STRING`	The design name. If not specified then root design name is used. This parameter is optional.

Examples

```
#command to find Electrical Constraint Sets on an XNet
cnsGetXNetConstraintSet db:00000013 ECSET
```

Related Commands

cnsGetXNetInDesign

cnsGetXNetInDesign

Gets a list of CNSOBJIDs for all the XNets in the design.

Return Type

List of CNSOBJIDs

Syntax

```
cnsGetXNetInDesign <design_name>
```

Parameters

Parameter	Type	Description
`design_name`	`STRING`	Name of the design This parameter is required.

Examples

```
#command to get XNets in the design root_des
cnsGetXNetInDesign root_des
```

Related Commands

cnsGetClassMemberXNet

createXNetPinPairDefinition

xnetPinPairdefinition

cnsGetDiffPairMemberXNet

cnsGetMatchGroupMemberXNet

cnsGetMemberXNet

cnsGetNetParentXNet

cnsGetXNetConstraint

cnsGetXNetConstraintSet

cnsGetXNetInDesign

cnsGetXNetMemberNet

cnsGetXNetMemberPinPair

cnsGetXNetParentBus

cnsGetXNetParentClass

cnsGetXNetParentDiffPair

cnsGetXNetParentMatchGroup

cnsGetXNetMemberNet

Returns a list of CNSOBJIDs of member nets of a specific XNet in the design.

Return Type

List of CNSOBJIDs

Syntax

```
cnsGetXNetMemberNet <xnetID> ?design_name?
```

Parameters

Parameter	Type	Description
`xnetID`	`CNSOBJID`	The XNet identifier This parameter is required.
`design_name`	`STRING`	The name of the design. If not specified then the root design name is used. This parameter is optional.

Examples

```
#command to return nets with CNSOBJID db:00000002 and db:00000008
#which are members of XNet db:00000015
cnsGetXNetMemberNet db:00000015
```

Related Commands

cnsGetClassMemberXNet

createXNetPinPairDefinition

xnetPinPairdefinition

cnsGetDiffPairMemberXNet

cnsGetMatchGroupMemberXNet

cnsGetMemberXNet

cnsGetNetParentXNet

cnsGetXNetConstraint

cnsGetXNetConstraintSet

cnsGetXNetInDesign

cnsGetXNetMemberPinPair

cnsGetXNetParentBus

cnsGetXNetParentClass

cnsGetXNetParentDiffPair

cnsGetXNetParentMatchGroup

cnsGetXNetMemberPinPair

Returns a list containing CNSOBJIDs of pin pair members of a specific XNet in the design.

Return Type

List of CNSOBJIDs

Syntax

```
cnsGetXNetMemberPinPair <xnetID> ?design_name?
```

Parameters

Parameter	Type	Description
`xnetID`	`CNSOBJID`	The XNet identifier. This parameter is required.
`design_name`	`STRING`	The name of the design. If not specified then root design name is used. This parameter is optional.

Examples

```
#command to get pin pairs of the XNet db:00000015 in the root design
cnsGetXNetMemberPinPair db:00000015
#command to get pin pairs of the XNet db:00000049 in the design memory
cnsGetXNetMemberPinPair db:00000049 memory
```

Related Commands

cnsGetClassMemberXNet

createXNetPinPairDefinition

xnetPinPairdefinition

cnsGetDiffPairMemberXNet

cnsGetMatchGroupMemberXNet

cnsGetMemberXNet

cnsGetNetParentXNet

cnsGetXNetConstraint

cnsGetXNetConstraintSet

cnsGetXNetInDesign

cnsGetXNetMemberNet

cnsGetXNetParentBus

cnsGetXNetParentClass

cnsGetXNetParentDiffPair

cnsGetXNetParentMatchGroup

cnsGetXNetParentBus

Returns the CNSOBJID of the parent bus of the given XNet

Return Type

CNSOBJID

Syntax

```
cnsGetXNetParentBus <xnetID> ?bTraverseHier? ?design_name?
```

Parameters

Parameter	Type	Description
`xnetID`	`CNSOBJID`	The XNet identifier This parameter is required.
`bTraverseHier`	`INT`	The parameter to set the traversal direction. This parameter is optional. Default value is `1`.
`design_name`	`STRING`	The name of the design. If not specified then root design name is used. This parameter is optional.

Examples

```
#command to get parent bus of the given XNet db:00000013 in the root design
cnsGetXNetParentBus db:00000013
#command to get parent bus of the given XNet db:00000113 in the design memory
cnsGetXNetParentBus db:00000113 memory
```

Related Commands

cnsGetClassMemberXNet

createXNetPinPairDefinition

xnetPinPairdefinition

cnsGetDiffPairMemberXNet

cnsGetMatchGroupMemberXNet

cnsGetMemberXNet

cnsGetNetParentXNet

cnsGetXNetConstraint

cnsGetXNetConstraintSet

cnsGetXNetInDesign

cnsGetXNetMemberNet

cnsGetXNetMemberPinPair

cnsGetXNetParentClass

cnsGetXNetParentDiffPair

cnsGetXNetParentMatchGroup

cnsGetXNetParentClass

Returns the CNSOBJIDs of parent net class of the specified XNet

Return Type

CNSOBJID

Syntax

```
cnsGetXNetParentClass <xnetID> ?bTraverseHier? ?design_name?
```

Parameters

Parameter	Type	Description
`xnetID`	`CNSOBJID`	The XNet identifier. This parameter is required.
`bTraverseHier`	`INT`	The parameter to set the traversal direction. This parameter is optional. Default value is `1`.
`design_name`	`STRING`	The name of the design. If not specified then root design name is used. This parameter is optional.

Examples

```
#command to get parent net class of the given XNet db:00000013 in the root design
cnsGetXNetParentClass db:00000013
#command to get parent net class of the given XNet db:00000113 in the design memory
cnsGetXNetParentClass db:00000113 memory
```

cnsGetXNetParentDiffPair

Returns the CNSOBJID of the parent differential pair of the specified XNet

Return Type

CNSOBJID

Syntax

```
cnsGetXNetParentDiffPair <xnetID> ?bTraverseHier? ?design_name?
```

Parameters

Parameter	Type	Description
`xnetID`	`CNSOBJID`	The XNet identifier. This parameter is required.
`bTraverseHier`	`INT`	The parameter to set the traversal direction. This parameter is optional. Default value is `1`.
`design_name`	`STRING`	The name of the design. If not specified then root design name is used. This parameter is optional.

Examples

```
#command to get parent differential pair of the given XNet db:00000013 in the root
design
cnsGetXNetParentDiffPair db:00000013
#command to get parent differential pair of the given XNet db:00000113 in the design
memory
cnsGetXNetParentDiffPair db:00000113 memory
```

cnsGetXNetParentMatchGroup

Returns a list containing the CNSOBJIDs of parent match groups of a specific XNet.

Return Type

List of CNSOBJIDs

Syntax

```
cnsGetXNetParentMatchGroup <xnetID> ?bTraverseHier? ?design_name?
```

Parameters

Parameter	Type	Description
`xnetID`	`CNSOBJID`	The XNet identifier. This parameter is required.
`bTraverseHier`	`INT`	The parameter to set the traversal direction. This parameter is optional. Default value is `1`.
`design_name`	`STRING`	The name of the design. If not specified then root design name is used. This parameter is optional.

Examples

```
#command to get a list of all parent match groups of the XNet db:00000013 in root
design
cnsGetXNetParentMatchGroup db:00000013
#command to get a list of all parent match groups of the XNet db:00000113 in design
memory
cnsGetXNetParentMatchGroup db:00000113 memory
```

Related Commands

cnsGetMatchGroupConstraint

cnsGetMatchGroupConstraintSet

cnsGetMatchGroupInDesign

cnsGetMatchGroupMemberNet

cnsGetMatchGroupMemberPinPair

cnsGetMatchGroupMemberXNet

cnsGetNetParentMatchGroup

cnsImportConstraintFile

Use cnsImportConstraintFile to import a constraints file into the design.

Return Type

INT

Syntax

```
cnsImportConstraintFile <designName> <filePath>
```

Parameters

Parameter	Type	Description
`designName`	`STRING`	Name of the design. This parameter is required.
`filePath`	`STRING`	Path of the constraint file. This parameter is required.

Examples

```
#command to import the constraints from the file /home/constraints/rpd.txt into the
design workshop
cnsImportConstraintFile workshop /home/constraints/rpd.txt
```

cnsImportECSet

Use cnsImportECSet to import the ECSet file. It returns 1 for success and 0 for failure.

Return Type

INT

Syntax

```
cnsImportECSet <ecSet_file_path> ?overwrite_mode? ?design_name?
```

Parameters

Parameter	Type	Description
`ecSet_file_path`	`STRING`	The path to the ECSet (.top) file. This parameter is required.
`overwrite_mode`	`INT`	The Overwrite mode. This parameter is optional. Default value is `1`.
`design_name`	`STRING`	Name of the design. If not specified then root design name is used. This parameter is optional.

Examples

```
#command to import the electrical constraint sets from the file
d:/sda_demo/results/SATA_TX_P_4.top
#into the root design
cnsImportECSet D:/sda_demo/results/SATA_TX_P_4.top
```

cnsImportTechFile

Use cnsImportTechFile to import a technology file, which is an ASCII file that is read into a design to specify user-preferred units, constraint and parameter values, and user properties. It returns 1 for success and 0 for failure.

Return Type

INT

Syntax

```
cnsImportTechFile <tech_file_path> ?overwrite_mode? ?design_name?
```

Parameters

Parameter	Type	Description
`tech_file_path`	`STRING`	Tech file path. This parameter is required.
`overwrite_mode`	`INT`	The Overwrite mode. This parameter is optional. Default value is `1`.
`design_name`	`STRING`	The name of the design. If not specified then root design name is used. This parameter is optional.

Examples

```
#command to import the technology file sample.tcfx into the root design
cnsImportTechFile D:\sample_design\results\sample.tcfx
```

cnsmgr

Opens the Constraint Manager window.

Return Type

NONE

Syntax

```
cnsmgr
```

Examples

```
#open the constraint manager window
cnsmgr
```

createXNetPinPairDefinition

Use createXNetPinPairDefinition to create an XNet pin pair definition on pins selected on the canvas. The pins must belong to the same component.

Return Type

NONE

Syntax

```
sch::createXNetPinPairDefinition
```

Examples

```
sch::createXNetPinPairDefinition
```

dbGetMemberNetNames

This command lists the names of all the members, such as wires, buses, and NetGroups of the selected NetGroup.

Return Type

LIST

Syntax

```
sch::dbGetMemberNetNames <dbID>
```

Parameters

Parameter	Type	Description
`<dbID>`	`DBID`	dbID of the NetGroup currently selected on the canvas This parameter is required.

Examples

```
#command to store the dbID of the selected NetGroup in a variable called netGrpDbID
#then shows the names of all the members of the selected NetGroup
set netGrpDbID [ sch::dbGetSelectedItems [ sch::dbGetActivePage ] ]
sch::dbGetMemberNetNames $netGrpDbID
#For Example: Net1 Net2
```

Related Commands

dbGetMemberNetSpaths

getGridSettings

Returns the grid settings in JSON format in the following format:
{
"Display Electrical Grid": "5",
"Documentation Grid": "0.1",
"Electrical Grid": "0.5",
"Grid Style": "Lines",
"Grid Units": "INCHES",
"Pin-to-Pin Spacing": "0.1"
}

Return Type

STRING

Syntax

```
sch::getGridSettings
```

Examples

```
sch::getGridSettings
```

importElectricalCsetsFileDialog

This command opens the Import Electrical CSets dialog.

Return Type

NONE

Syntax

```
importElectricalCsetsFileDialog
```

Examples

```
#command to open the dialog for selecting an Electrical Constraint Sets file for import
importElectricalCsetsFileDialog
```

importPinDelayFileDialog

Opens the Import Pin Delay file dialog box.

Return Type

NONE

Syntax

```
importPinDelayFileDialog
```

Examples

```
#comamnd to open the file dialog for importing a pin delay file
importPinDelayFileDialog
```

netNavigationDump

It dumps the connected nets data for a navigated net to a file.

Return Type

NONE

Syntax

```
netNavigationDump <indentor> <path>
```

Parameters

Parameter	Type	Description
`<indentor>`	`STRING`	A character which will represent a column gap This parameter is required.
`<path>`	`STRING`	The file path where navigation data is to be dumped This parameter is required.

Examples

```
netNavigationDump - ../netNavigationDump.csv
```

Related Commands

pinNavigationDump

pinNavigationDump

It dumps the connected pins data for a navigated net to a file.

Return Type

NONE

Syntax

```
pinNavigationDump <indentor> ?path?
```

Parameters

Parameter	Type	Description
`indentor`	`STRING`	A character which will represent a column gap This parameter is required.
`path`	`STRING`	The file path where navigation data is to be dumped This parameter is required.

Examples

```
pinNavigationDump - ../pinNavigationDump.csv
```

Related Commands

netNavigationDump

runPartMgr

This command will show the Part Manager dialog box with the status of the parts in the currently open design for the parts present in the reference library. The status of the part could be In sync, Auto sync, or Manual sync.

Return Type

NONE

Syntax

```
runPartMgr
```

Examples

```
runPartMgr
```

Related Commands

autoupdateparts

setConstraint

setConstraint command is used to assign constraints to nets, xnets, net class objects, and match groups.
By default, the command works on the object(s) selected on canvas. To run the command on a different object the object can be specified as
a. XNet - use -xnet <xnet name>
b. Net Class - use -netclass <class name>
c. Bus - use -bus <bus name>
d. Differential pair - use -dp <differential pair name>
e. Net Group - use -ng <net group name>

It supports several different options for different constraints.

1. setConstraint -simple [list <name> <value>]
The above is used to set simple name = value type of constraints where name can be any of the below:
RATSNEST_SCHEDULE (valid values are MIN_DAISY_CHAIN, MIN_TREE, SOURCE_LOAD_DAISY_CHAIN, STAR, FAR_END_CLUSTER)
NET_SCHEDULE (valid values are VERIFY, DO_NOT_VERIFY)
STUB_LENGTH (numeric value)
MAX_EXPOSED_LENGTH (numeric value)
MAX_VIA_COUNT (numeric value)
TOTAL_ETCH_LENGTH_MIN (numeric value)
TOTAL_ETCH_LENGTH_MAX (numeric value)
Each of the above can be unset by passing empty quotes as the value.

2. setConstraint -NETCLASS [list <type of net class> <name of net class> ?-del?]
This adds the object specified (or the selected net on canvas) to an existing electrical, spacing or physical net class.
If the electrical net class does not exist, first it gets created and then the object gets added to it.
Valid values for type of net class are electrical/physical/spacing
To remove the object from the net class use the -del option

3. setConstraint -ECSET|PCSET|SCSET -name <name of constraint set>| -del
This adds the object to an electrical (ECSET), physical (PCSET) or spacing (SCSET) constraint set.
In order to remove the object from the constraint set use -del in place of the name of the constraint set

4. setConstraint -propagation_delay [list -pin1 <pin name> -pin2 <pin name> -delay1 <delay value> -del1units <delay units> -delay2 0 -del2units mm -mindelay <min delay> -mindelayunits mm -

maxdelay <maximum delay> -maxdelayunits mm
This creates a min/max pin pair between pin1 and pin2 with minimum delay as <min delay> and maximum delay as <maximum delay>
Pin name 1 and 2 can also be one of AD:AR, L:S or D:R

5. setConstraint -matchgroup [list -name {<match_group_name>} -pin1 <refdes.pin_number> -pin2 <refdes.pin_number> -scope {local/global} -delta <delta value> -deltaunits <delta units> -tolerance <tolerance value> -tolunits <tolerance units>
This creates a match group with name <match_group_name> and adds the selected net to the match group with relative propagation delay defined on the pin pair specified by pin1 and pin2, scope as local or global, and the given delta and tolerance values

Return Type

INT

Syntax

```
?-xnet <xnet_name> | -class <net_class_name> | -bus <bus_name> | -dp
<differential_pair_name> | -ng <netgroup_name>? -simple [list <constraint_name>
<constraint_value>] | -netclass [list electrical|physical|spacing <class_name> ?-del?]
| -matchgroup [list -name {<match_group_name>} -pin1 <refdes.pin_number> -pin2
<refdes.pin_number> -scope {local/global} -delta <delta value> -deltaunits <delta
units> -tolerance <tolerance value> -tolunits <tolerance units>] | -propagation_delay
[list -pin1 <refdes.pin_number> -pin2 <refdes.pin_number> -delay1 <delay> -del1units
<units> -delay2 <delay> -del2units <units> -mindelay <minimum_delay_value> -
mindelayunits <minimum_delay_units> -maxdelay <maximum_delay_value> -maxdelayunits
<maximum_delay_units>]
```

Parameters

Parameter	Type	Description
`-xnet <xnet_name>`	`STRING`	The name of the XNet on which constraints are to be captured. This parameter is optional.

`-class <net_class_name>`	`STRING`	The name of an electrical net class on which constraints are to be captured. This parameter is optional.
`-bus <bus_name>`	`STRING`	The name of the bus on which contstraints are to be captured This parameter is optional.
`-dp <differential_pair_name>`	`STRING`	The name of the differential pair on which constraints are to be captured This parameter is optional.
`-ng <net_group_name>`	`STRING`	The name of the NetGroup on which constraints are to be captured This parameter is optional.
`-simple [list <constrain_name> <constraint_value>]`	`STRING`	Used for capturing basic constraints like STUB_LENGTH and VIA_COUNT which are name=value pairs. This parameter is optional.
`-netclass [list electrical|physical|spacing <class_name> ?-del?]`	`STRING`	Used to assign a net class constraint. If 'electrical' is specified and the net class does not exist, the net class is created as well. -del option can be used to remove the net from the net class This parameter is optional.
`-matchgroup [list -name {<match_group_name>} -pin1 <refdes.pin_number> -pin2 <refdes.pin_number> -scope {local/global} -delta <delta value> -deltaunits <delta units> -tolerance <tolerance value> -tolunits <tolerance units>`	`STRING`	Creates a match group and adds the net to the match group. Delta and tolerance constraints are captured on the pin pair of the net where pin1 is the driver pin and pin2 is the receiver pin This parameter is optional.

`-propagation_delay`	`STRING`	Used to create or edit pin pair constraints. This parameter is optional.
`-pin1 <refdes.pin_number>`	`STRING`	Driver pin of the pin pair. This parameter is required.
`-pin2 <refdes.pin_number>`	`STRING`	The receiver pin of the pin pair. The value is reference designator followed by a dot followed by the pin number. For All Driver/All Receiver it will be AR, for Longest/Shortest it will be S, and for Longest/Shortest Driver Receiver it will be R This parameter is required.
`-del1units <units>`	`STRING`	Units for delay for driver pin of the pin pair. This parameter is optional.
`-del2units <units>`	`STRING`	Units for delay for receiver pin of the pin pair. This parameter is optional.
`-mindelay <minimum_delay_value>`	`STRING`	Minimum delay for the pin pair This parameter is optional.
`-mindelayunits <minimum_delay_units>`	`STRING`	Units for the minimum delay of the pin pair This parameter is optional.

Examples

```
#Example 1: add net group TESTNG to electrical net class NGCLASS
setConstraint -ng "TESTNG" -NETCLASS [list electrical NGCLASS ]

#Example 2: create a matchgroup based relative propagation delay pin pair between pins
U1.12 and R1.2 with scope Global and tolerance 5%
setConstraint -MATCHGROUP [list -name C1_M1 -pin1 U1.12 -pin2 R1.2 -scope Global -delta
0 -deltaunits ns -tolerance 5 -tolunits %]

#Example 3: set min-max propagation delay on the selected net
setConstraint -propagation_delay [list -pin1 AD -pin2 AR -delay1 0 -del1units mm -
delay2 0 -del2units mm -mindelay 1 -mindelayunits mm -maxdelay 2 -maxdelayunits mm]

#Example 4: set STUB_LENGTH as 10 for XNET ADDR_X
setConstraint -xnet ADDR_X -simple [list STUB_LENGTH 10]
```

Related Commands

extractEcset

validateNets

This command validates all the nets in the design. XNets are updated as per design connectivity and their member objects, such as buses, net groups, differential pairs, and net classes, get updated.

Return Type

NONE

Syntax

```
sch::validateNets <blockName>
```

Parameters

Parameter	Type	Description
`<blockName>`	`STRING`	Name of the design for which net validation is to be carried out This parameter is required.

Examples

```
sch::validateNets root_design
```

xnetPinPairdefinition

This command is used to add, delete, copy or reset xnet pin-pair definitions.
1. Add -
xnetPinPairdefinition -add. This command works when 2 pins of an instance are pre-selected on canvas and will add XNET_PINS property on selected pins on canvas.
xnetPinPairdefinition -add -pins <list_of_pin_pairs>. This command works when an instance is pre-selected on canvas and will add XNET_PINS property on pin pairs specified in the list.

2. Delete -
xnetPinPairdefinition -del <pin_pair> <XNetNum>. This command works when an instance is pre-selected on canvas and will delete XNET_PINS property with value XNetNum on pin pair specified.

3. Copy -
xnetPinPairdefinition -source <source_spath> -copyTo [list <dest_spath>] . This command works when an instance is pre-selected on canvas and will copy XNET_PINS property definitions from instance with spath source_spath to instances with spath values specified in copyTo list.

4. Reset -
xnetPinPairdefinition -reset [list <obj_spath>]. This command works when an instance is pre-selected on canvas and will reset XNET_PINS property definitions from instances specified in the list.

Return Type

NONE

Syntax

```
xnetPinPairdefinition -add ?-pins <list_of_pin_pairs>? | -del <pin_pair> <XNetNum> | -
source <source_spath> -copyTo [list <dest_spath>] | -reset [list <obj_spath>]
```

Parameters

Parameter	Type	Description
`list_of_pin_pairs`	`LIST`	Used with -add. Used to apply pin pair definitions to the selected instance. Syntax is [list <pin1>,<pin2> <pin3>,<pin4> ... <pinN-1>,<pinN>]. Note that the pins in a pair are separated by a comma, and the pin pairs are separated by a space. For example: xnetPinPairdefinition -add -pins [list inb,outy ina,gnd] This parameter is required.
`pin_pair`	`LIST`	Used with -del. Used to delete pin pair definition from the selected instance. Syntax is [list <pin1> <pin2>]. Note that the pins in the pair are separated by a space. For example: xnetPinPairdefinition -delete [list inb outy] 1 This parameter is optional.
`XNetNum`	`INTEGER`	Used with -del. Used to specify XNET_PINS property value of pin pair on which xnet definition has to be deleted. This parameter is required.
`source_spath`	`LIST`	Used with -source. Used to specify spath of instance from which xnet pin pair definitions are to be copied. This parameter is required.
`dest_spath`	`LIST`	Used with -source. Used to specify spaths of instances on which xnet pin pair definitions are to be copied to This parameter is required.
`obj_spath`	`LIST`	User with -reset. Specify spaths of instances on which xnet pin pair definitions are to be reset This parameter is required.

Examples

```
#Examples for adding xnet.
xnetPinPairdefinition -add
xnetPinPairdefinition -add -pins [list inb,outy ina,gnd]

#Delete -
xnetPinPairdefinition -delete [list inb outy ] 1

#Copy -
xnetPinPairdefinition -source "@worklib.root(tbl_1):\\I7\\" -copyTo [list
"@worklib.root(tbl_1):\\I2\\@worklib.usb3(tbl_1):\\I1\\"
"@worklib.root(tbl_1):\\I1\\@worklib.pci_blk(tbl_1):\\I1\\"
"@worklib.root(tbl_1):\\I8\\" "@worklib.root(tbl_1):\\I2\\@worklib.usb3(tbl_1):\\I7\\"
]

#Reset -
xnetPinPairdefinition -reset [list
"@worklib.root(tbl_1):\\I1\\@worklib.pci_blk(tbl_1):\\I2\\"
"@worklib.root(tbl_1):\\I1\\@worklib.pci_blk(tbl_1):\\I4\\" ]
```

3

Graphics Database Query

areAllSelected

Returns true if all items in the Selection filter are selected or else returns false.

Return Type

String

Syntax

```
schSelectionFilterUtils::areAllSelected
```

dbConvertToDBUnits

Converts x-y coordinates in user units to database units. Returns 0 0 if user coordinates are invalid. sch::dbGet family of commands return coordinate values in database units.

Return Type

POINT

Syntax

```
sch::dbConvertToDBUnits <userCoordinates>
```

Parameters

Parameter	Type	Description
`userCoordinates`	`POINT`	Coordinates in user units This parameter is required.

Examples

```
#Command to convert the specified grid units to database units
#User-specified coordinates are converted to their corresponding database coordinate
values
sch::dbConvertToDBUnits {6702 9976}
#Output: 1702308 2533904
```

Related Commands

dbConvertToUserUnits

dbConvertToUserUnits

It converts the coordinates from the database unit to the grid unit. It returns 0 0 if database units are invalid. All the sch::dbGet commands have the output value in database units.

Return Type

POINT

Syntax

```
sch::dbConvertToUserUnits <point_coordinates>
```

Parameters

Parameter	Type	Description
`point_coordinates`	`POINT`	coordinates of database unit This parameter is required.

Examples

```
#command to convert the specified database units to user units/grid units
#Database coordinates are converted to corresponding user coordinate values
#Output of position of a selected item using dbGetPos will be in database units
sch::dbGetPos [ sch::dbGetSelectedItems [ sch::dbGetActivePage ] ]
#Output: 3086100 2755900
#It can be converted into grid units using dbConvertToUserUnits
sch::dbConvertToUserUnits [ sch::dbGetPos [ sch::dbGetSelectedItems [
sch::dbGetActivePage ] ] ]
#Output: 12150 10850
```

Related Commands

dbConvertToDBUnits

dbGetPos

dbGetSelectedItems

dbGetActivePage

dbCreatePageSPath

Returns the sPath of the page for the given library, cell, and page.

Return Type

STRING

Syntax

```
sch::dbCreatePageSPath <library_name> <cell_name> <page_name>
```

Parameters

Parameter	Type	Description
`library_name`	`STRING`	Name of the library This parameter is required.
`cell_name`	`STRING`	Name of the cell This parameter is required.
`page_name`	`STRING`	Name of the page This parameter is required.

Examples

```
#command to store the library name using the following command in a variable called
libraryName
set libraryName [ cps::getDirectiveValue GLOBAL def_worklib ]
#command to store the cell name using the following command in a variable called
cellName
set cellNames [ sch::dbGetCellNames $libraryName ]
#command to store the view name using following command in a variable called viewName
set pageNames [ sch::dbGetPageNames $libraryName [lindex $cellNames 0] ]
sch::dbCreatePageSPath $libraryName [lindex $cellNames 0] [lindex $pageNames 0]
#Output: @worklib.des(tbl_1):Root_Page(1)
```

dbFindProperty

Returns the value of the specified property on an CCanonicalPathContainer which is an object having valid sPath.

Return Type

STRING

Syntax

```
sch::dbFindProperty <CCanonicalPathContainer> <property_name>
```

Parameters

Parameter	Type	Description
`CCanonicalPathContainer`	`CCanonicalPathContainer`	CCanonicalPathContainer object This parameter is required.
`property_name`	`STRING`	Name of property to find on the object This parameter is required.

Examples

```
package require sch 1.0
catch { load $env(ADE_CONNSERVER_LIB) TddConnectivityServer }

proc run_me {} {
set c [getServer]
set selectedItem [sch::dbGetSelectedItems [sch::dbGetActivePage]]
set spath [ sch::dbGetSPath $selectedItem ]
#pCont is CCanonicalPathContainerObject
set pCont [ $c findFromSpath $spath ]
#used it for a Net object which has PHYS_NET_NAME property
set prop [ sch::dbFindProperty $pCont "PHYS_NET_NAME" ]
puts "propVal = $prop"
delete_CCanonicalPathContainer $pCont
}
```

dbGetActivePage

Returns the database identifier for the current page.

Return Type

DBID

Syntax

```
sch::dbGetActivePage
```

Examples

```
#command to return the DBID for the active page
#if db:00000016 is the db id of the current page
sch::dbGetActivePage
#Output: db:00000016
```

Related Commands

dbGetActivePageSpath

dbGetPage

dbGetPageItems

dbGetPageNames

dbGetPageSizeType

dbGetPagesOfComponent

dbGetPagesOfNet

dbGetPageSummary

dbGetActivePageSpath

Returns the sPath (internal path) of the active page. It can be used as a sub-command for other commands, such as openItem, where the page spath is required to open a page.

Return Type

SPATH

Syntax

```
sch::dbGetActivePageSpath
```

Examples

```
#command to get sPath for active page
sch::dbGetActivePageSpath
#Output: @worklib.block1(tbl_1):page(1)
#command to open page(1) and make it the active page
set page [ sch::dbGetActivePageSpath ] #(save current page sPath in variable page)
#now go to any other page and run
openItem $page SCH PAGE #(this will open page and make it active again)
```

Related Commands

dbGetActivePage

dbGetPage

dbGetPageItems

dbGetPageNames

dbGetPageSizeType

dbGetPagesOfComponent

dbGetPagesOfNet

dbGetPageSummary

dbGetBBox

Returns the bounding box of the object. It can be used as a sub-command for many other commands.

This function returns the union of the bounding box of the object, its properties, and other text items associated with it. Since text rendering is font-engine and display-dependent (DPI, physical screen size, display resolution, text scaling, and so on), the bounding box dimensions of text objects may vary if the screen/display attributes change. Some of the ways such changes can occur are if the application window is moved from one monitor to another, if the display resolution is changed, or if the text scaling is modified.

Also, this function returns the bounding box information by rendering the items on the screen, a change in the display parameter might lead to differences between the values returned before and after an operation for the same object.

To avoid such issues, use the dbGetShapeBBox functions for performing geometry-sensitive computation on non-text objects.

Return Type

BBOX

Syntax

```
sch::dbGetBBox <item_id>
```

Parameters

Parameter	Type	Description
`item_id`	`STRING`	dbID of the item This parameter is required.

Examples

```
#command to return BBOX of the item which has DBID (db:0000001c)
sch::dbGetBBox db:0000001c
#output: {3335866 3810000} {3649133 4869180}
#The first set of coordinates corresponds to the top-left corner of the bounding box,
#and the second set of coordinates corresponds to the bottom-right corner of the
#bounding box
#Now, to change line style for the selected item
setLineType [sch::dbGetBBox [ sch::dbGetSelectedItems [ sch::dbGetActivePage ] ] ] dot
#this command will take active page's selected item, extract its BBOX and apply dot
line style to it.
```

Related Commands

dbGetItemsInBBox

dbGetItemsInBBoxByType

dbGetLibCellViewBBox

dbGetShapeBBox

dbGetSelectedItems

dbGetActivePage

dbGetBlockInstanceName

It returns the instance name of a block in the design based on the sPath provided.

Return Type

STRING

Syntax

```
sch::dbGetBlockInstanceName <block_spath>
```

Parameters

Parameter	Type	Description
`block_spath`	`STRING`	sPath of the block This parameter is required.

Examples

```
#command to get the instance name of the block
#find the sPath of the block and use this command on it
sch::dbGetSPath [ sch::dbGetSelectedItems [ sch::dbGetActivePage ] ]
#the sPath of the selected block is returned
#@worklib.power(tbl_1):\I150\
#use the following sPath to find the instance name of the block
sch::dbGetBlockInstanceName [ sch::dbGetSPath [ sch::dbGetSelectedItems [
sch::dbGetActivePage ] ] ]
#Output: new_block(i143)
```

Related Commands

dbGetBlockSPath

dbGetSPath

dbGetSelectedItems

dbGetActivePage

dbGetBlockPins

Returns a list of DBIDs of the pins in a block. The block can be a hierarchical block or system block.

Return Type

LIST

Syntax

```
sch::dbGetBlockPins <itemID>
```

Parameters

Parameter	Type	Description
`itemID`	`STRING`	Block ID for the selected hierarchical block or system block This parameter is required.

Examples

```
# This command will return the itemId of the selected item
set itemId [ sch::dbGetSelectedItems [ sch::dbGetActivePage ]]
puts $itemId
# following is an example of itemId of selected object
# db:0000001f

# All the system block pin IDs for the system block will be returned.
# Only one pin is present in the functional block, so only one ID is displayed.
sch::dbGetBlockPins $itemId
# Above command will return following output
# db:00000018
```

Related Commands

dbGetSelectedItems

dbGetActivePage

dbGetBlockSPath

It returns the sPath of the design name given. It can be used as a sub-command for many other commands.

Return Type

SPATH

Syntax

```
sch::dbGetBlockSPath <design_name>
```

Parameters

Parameter	Type	Description
`design_name`	`STRING`	name of the design or block This parameter is required.

Examples

```
#The design or cell name of the block is given as a parameter and
#its sPath is returned
sch::dbGetBlockSPath workshop1
#Output: @worklib.workshop1(tbl_1)
#the command below will give properties of the selected block
sch::dbGetProperties [ sch::dbGetSelectedItems [ sch::dbGetActivePage ] ] [
sch::dbGetBlockSPath block_name ]
#{PN 81-0000445-01 0 0 String 0 0} {SYMBOL_OUTLINE -250,50,250,-2000 0 0 String 0 0}
#{PART_NAME ABC8272_3M_XYZ58-81-0000445-01A 0 0 String 0 0}
#{CHIPS_PART_NAME ABC8272_3M_NOP58 0 0 String 0 0}
#{JEDEC_TYPE ABC58_1800X2300X750 0 0 String 0 0} {LOCATION U1 0 1 String 0 0}
#{MANUFACTURER ABCD 0 0 String 0 0} {MANUFACTURER_PN ABC8273MNOPQ 0 1 String 0 0}
#{PART_NAME ABC8272_3M_NOP58 0 0 String 0 0} 0 0 String 0 0}
```

Related Commands

dbGetProperties

dbGetSelectedItems

dbGetActivePage

dbGetCellNames

It returns a list of all the cell names separated by space present in any library accessible to the project. It will return an empty string (" ") if no cell is found in that library.

Return Type

LIST

Syntax

```
sch::dbGetCellNames <library_name>
```

Parameters

Parameter	Type	Description
`library_name`	`STRING`	Name of the library from which cells will be listed This parameter is required.

Examples

```
#command to return all the cell names in the library
sch::dbGetCellNames discretes
#output: zener_a2k1 zener_a1k2 tpd2eusb30 tce0806g#2d900#2d2p stps3045djf_pflat5x6
#sir412dp si1411dh schottky_a2k1 schottky_a1k2 schottky_a1a2k3 schottkya1a2k3
#pnp_bce pmos_gds nup2201mr6t1g nup2201 nmos_gds mmbt3904 ltv_817s
#led_rgb led_dual_red_grn_5710112f led_dual_red_5710111f
```

Related Commands

dbGetLibCellView

dbGetLibCellViewBBox.

dbGetChildren

It returns a list of all the children for the given item. Children consists of all the properties and other related objects of the given item.

Return Type

LIST

Syntax

```
sch::dbGetChildren <item_id>
```

Parameters

Parameter	Type	Description
`item_id`	`DBID`	ID of the item This parameter is required.

Examples

```
#the given id has 5 children. So command will return 5 ids
sch::dbGetChildren db:00000022
#output: db:00000021 db:00000020 db:0000001f db:0000001e db:0000001d
#select any item and run the below command to get children of selected item.
sch::dbGetChildren [ sch::dbGetSelectedItems [ sch::dbGetActivePage ] ]
#output: db:00000038 db:00000039 db:0000003a db:0000003b db:0000003c db:0000003d
#db:0000003e db:0000003f db:00000040 db:00000041 db:00000042 db:00000043 db:00000044
#db:00000045 db:00000046 db:00000047 db:00000048 db:00000049 db:0000004a db:0000004b
#db:0000004c db:0000004d
```

Related Commands

dbGetSelectedItems

dbGetActivePage

dbGetConnectedItems

It returns the database identifiers (dbID) of all the connected routes to the pin with dbId <pinId>. If no route is connected to the pin, nothing is returned.

Return Type

LIST

Syntax

```
sch::dbGetConnectedItems <pinId>
```

Parameters

Parameter	Type	Description
`<pinId>`	`STRING`	DBID of the pin This parameter is required.

Examples

```
#the following example assumes that you have some pins selected on your canvas
#command to get the currently active page
set currentActivePage [sch::dbGetActivePage]
#command to get the currently selected pins
set currentlySelectedPins [sch::dbGetSelectedItems $currentActivePage]
#command to get the dbId for each connected route to the pin and print it
foreach item $currentlySelectedPins { set connectedRouteList [sch::dbGetConnectedItems
$item]; puts "Route ids connected to $item are $connectedRouteList "}
```

dbGetCSPropNameVal

It returns the list of displayed properties and values of an object.

Return Type

STRING

Syntax

```
sch::dbGetCSPropNameVal <dbId>
```

Parameters

Parameter	Type	Description
`<dbId>`	`STRING`	dbID of an item This parameter is required.

Examples

```
#command to get the currently active page
set currentActivePage [sch::dbGetActivePage]
#command to get the currently selected items
set currentlySelectedItem [sch::dbGetSelectedItems $currentActivePage]
#loop over the dbIds for the currently selected items, get the property list
#and print it out
foreach item $currentlySelectedItem { set propList[sch::dbGetCSPropNameVal $item]; puts
"Property list for item $item is $propList"}
```

dbGetDesignPages

Return information about all occurrences of pages in a design as a JSON string.

Return Type

STRING

Syntax

```
sch::dbGetDesignPages
```

Examples

```
# Sample Tcl script to query information from JSON
package require json

set pagesJSON [ sch::dbGetDesignPages]
set pagesDict [ json::json2dict $pagesJSON]

foreach page [ dict get $pagesDict pages] {
set pageObj [ dict create {*}$page ]
puts "Page id: [ dict get $pageObj id]"
puts "Page name: [ dict get $pageObj name]"
}
```

dbGetElectricalParents

It returns the electrical parents of the selected item, such as a dictionary of key-value pairs. For any route, it returns one of the following types of electrical parents: XNetParent, DiffPairParent, NetGroupParent, NetClassParent, MatchGroupParent.

Return Type

DICT

Syntax

```
sch::dbGetElectricalParents <routeId>
```

Parameters

Parameter	Type	Description
`<routeId>`	`STRING`	routeId of the component This parameter is required.

Examples

```
#command to get the currently active page
set currentActivePage [sch::dbGetActivePage]
#command to get the current selected items
set currentlySelectedItem [sch::dbGetSelectedItems $currentActivePage]
#loop over the dbIds for the currently selected items, get the electrical parents
#and print it out
foreach item $currentlySelectedItem { set elecParents [sch::dbGetElectricalParents
$item]; puts "Parents for item $item is $elecParents"}
```

dbGetFillColor

It returns the fill color of an item in the hexadecimal notation. For example, it will return 0x000000 as a hexadecimal notation for black.

Return Type

STRING

Syntax

```
sch::dbGetFillColor <dbId>
```

Parameters

Parameter	Type	Description
`<dbId>`	`STRING`	DBID of the component This parameter is required.

Examples

```
#command to get the current active page
set currentActivePage [sch::dbGetActivePage]
#command to get the current selected items
set currentlySelectedItem [sch::dbGetSelectedItems $currentActivePage]
#loop over the dbIds for the currently selected items, get the fill color
#and print it out
foreach item $currentlySelectedItem { set fillColor [sch::dbGetFillColor $item]; puts
"Fill Color for item $item is $fillColor"}
```

Related Commands

dbGetFillStyle

dbGetFillStyle

It returns the fill style of the item. It can be either solid or none.

Return Type

STRING

Syntax

```
sch::dbGetFillStyle <dbId>
```

Parameters

Parameter	Type	Description
`dbId`	`STRING`	dbID of the component This parameter is required.

Examples

```
#command to get the current active page
set currentActivePage [sch::dbGetActivePage]
#command to get the current selected items
set currentlySelectedItem [sch::dbGetSelectedItems $currentActivePage]
#loop over the dbIds for the currently selected items, get the fill style
#and print it out
foreach item $currentlySelectedItem { set fillStyle [sch::dbGetFillStyle $item]; puts
"Fill Style for item $item is $fillStyle"}
```

Related Commands

dbGetFillColor

dbGetFontColor

It returns the color of the text with db identifier <dbId> in the hexadecimal notation. For example, it will return 0x000000 as a hexadecimal notation for black.

Return Type

STRING

Syntax

```
sch::dbGetFontColor <dbId>
```

Parameters

Parameter	Type	Description
`dbId`	`STRING`	DBID of the text This parameter is required.

Examples

```
#command to get the currently active page
set currentActivePage [sch::dbGetActivePage]
#command to get the currently selected text
set currentlySelectedText [sch::dbGetSelectedItems $currentActivePage]
#loop over the dbId for each selected text, get the font color and print it
foreach item $currentlySelectedText { set fontColor [sch::dbGetFontColor $item]; puts
"Font color for $item is $fontColor"}
```

Related Commands

dbGetFontSize

dbGetFontName

dbGetFontName

It returns the font name of the text with db identifier <dbId>.

Return Type

STRING

Syntax

```
sch::dbGetFontName <dbId>
```

Parameters

Parameter	Type	Description
`dbId`	`STRING`	dbID of the text This parameter is required.

Examples

```
#command to get the currently active page
set currentActivePage [sch::dbGetActivePage]
#command to get the currently selected text
set currentlySelectedText [sch::dbGetSelectedItems $currentActivePage]
#loop over the dbId for each selected text, get the font name and print it
foreach item $currentlySelectedText { set fontName [sch::dbGetFontName $item]; puts
"Font name for $item is $fontName "}
```

Related Commands

dbGetFontColor

dbGetFontSize

dbGetFontSize

It returns the font size of the text with db identifier <dbId>.

Return Type

INT

Syntax

```
sch::dbGetFontSize <dbId>
```

Parameters

Parameter	Type	Description
`dbId`	`STRING`	dbID of the text This parameter is required.

Examples

```
#command to get the currently active page
set currentActivePage [sch::dbGetActivePage]
#command to get the currently selected text
set currentlySelectedText [sch::dbGetSelectedItems $currentActivePage]
#loop over the dbId for each selected text, get the font size and print it
foreach item $currentlySelectedText { set fontSize [sch::dbGetFontSize $item]; puts
"Font size for $item is $fontSize "}
```

Related Commands

dbGetFontColor

dbGetFontName

dbGetHotSpot

It returns the coordinates of the hotspot of the pin with db identifier <pinId>. The returned hotspot coordinates are in terms of db-coordinates.

Return Type

STRING

Syntax

```
sch::dbGetHotSpot <pinId>
```

Parameters

Parameter	Type	Description
`pinId`	`STRING`	dbID of the pin This parameter is required.

Examples

```
#command to get the currently active page
set currentActivePage [sch::dbGetActivePage]
#command to get the currently selected pins
set currentlySelectedPins [sch::dbGetSelectedItems $currentActivePage]
#loop over the dbIds for the currently selected items, get the fill color
#and print it out
foreach item $currentlySelectedPins { set hotSpot [sch::dbGetHotSpot $item]; puts
"HotSpot for pin $item is $hotSpot"}
```

Related Commands

dbConvertToUserUnits

dbGetImage

Use dbGetImage to extract the specified image from the design and save it to a specified path.
Following message strings are generated in case of success and failure respectively:
Success: "Image is copied to <path>"
Failure: "Failed to get the image."

Return Type

STRING

Syntax

```
sch::dbGetImage <itemId> <path>
```

Parameters

Parameter	Type	Description
`itemId`	`DBID`	dbID of the image. This parameter is required.
`path`	`STRING`	Absolute path to the location where image output is to be saved. This parameter is required.

Examples

```
#command to save the selected image on current active page at c:\test
set path {c:\test}
set itemID [ sch::dbGetSelectedItems [ sch::dbGetActivePage ]]
sch::dbGetImage itemID path
```

Related Commands

dbGetSelectedItems

dbGetActivePage

dbGetItemName

It returns the name of the object for the corresponding dbID. It returns nothing if the dbID is incorrect.

Return Type

STRING

Syntax

```
sch::dbGetItemName <item_id>
```

Parameters

Parameter	Type	Description
`item_id`	`DBID`	dbID of the design object, such as instance, pin, or a route. This parameter is required.

Examples

```
#command to get the name of the selected items in the current page
sch::dbGetItemName [ sch::dbGetSelectedItems [ sch::dbGetActivePage ]]
```

Related Commands

dbGetSelectedItems

dbGetActivePage

dbGetItemText

dbGetItemsInBBox

It returns the list of database identifiers for the objects that are present in the specified bounding box. It will return "invalid command name" in case any parameter value is incorrect.

Return Type

LIST

Syntax

```
sch::dbGetItemsInBBox <page_id> <bbox_item>
```

Parameters

Parameter	Type	Description
`page_id`	`DBID`	dbID of the page. This parameter is required.
`bbox_item`	`BBOX`	Bounding box coordinates. This parameter is required.

Examples

```
sch::dbGetItemsInBBox [ sch::dbGetActivePage ] [sch::dbGetBBox [sch::dbGetSelectedItems
[ sch::dbGetActivePage ]]]
sch::dbGetItemsInBBox [ sch::dbGetActivePage ] {{0 0} {3748280 4357033}}
#second parameter in the command is a bounding box as mentioned in the example
#output:
#db:00000018 db:00000019 db:0000001a db:0000001f db:00000020 db:0000001b db:0000001c
#db:00000021 db:00000022 db:00000023 db:00000024 db:00000025 db:00000026 db:00000027
#db:00000028 db:00000029 db:0000001d db:0000001e db:0000002a db:0000002b db:0000002c
#db:0000002d db:0000002e db:0000002f db:00000017 db:00000030 db:00000031
```

Related Commands

dbGetActivePage

dbGetBBox

dbGetSelectedItems

dbGetItemsInBBoxByType

dbGetItemsInBBoxByType

It returns the list of dbID of the specified object types within a bounding box. It returns an empty string in case there are no matching items.

Return Type

STRING

Syntax

```
sch::dbGetItemsInBBoxByType <page_id> <bbox> <item_type>
```

Parameters

Parameter	Type	Description
`page_id`	`DBID`	dbID of the page This parameter is required.
`bbox`	`BBOX`	Bounding box coordinates. This includes coordinates for the top-left and bottom-right corners of the bounding box. This parameter is required.
`item_type`	`DBTYPE`	dbType is the type of db item This parameter is required.

Examples

```
sch::dbGetItemsInBBoxByType db:00000016 {{2661920 3170766} {2748280 3357033}} 16
sch::dbGetItemsInBBoxByType [ sch::dbGetActivePage ] [ sch::dbGetBBox
[sch::dbGetSelectedItems [ sch::dbGetActivePage ]]] [ sch::dbGetType [
sch::dbGetSelectedItems [ sch::dbGetActivePage ]]]
#Output:
#db:00000017
```

Related Commands

dbGetActivePage

dbGetBBox

dbGetSelectedItems

dbGetType

dbGetItemsInBBox

dbGetItemText

It returns the string text of a note item.

Return Type

STRING

Syntax

```
sch::dbGetItemText <dbId>
```

Parameters

Parameter	Type	Description
`dbId`	`DBID`	dbID of the note item. This parameter is required.

Examples

```
sch::dbGetItemText db:00000032
sch::dbGetItemText [sch::dbGetSelectedItems [ sch::dbGetActivePage ]]
#output:
#sample note
```

Related Commands

dbGetSelectedItems

dbGetActivePage

dbGetItemName

dbGetLibCellView

It returns the library name, cell name, and the view of the specified instance. It returns an empty string if dbID does not correspond to any instance.

Return Type

STRING

Syntax

```
sch::dbGetLibCellView <dbId>
```

Parameters

Parameter	Type	Description
`dbId`	`DBID`	dbID of the instance for which lib:cell:view is to be extracted. This parameter is required.

Examples

```
sch::dbGetLibCellView db:00000017
sch::dbGetLibCellView [sch::dbGetSelectedItems [ sch::dbGetActivePage ]]
#output:
#{cell cap} {library discrete} {view sym_1}
```

Related Commands

dbGetSelectedItems

dbGetActivePage

dbGetLineCapStyle

It returns the line cap style for the given dbID. Possible return values are square-cap, flat-cap, and round-cap. It will return an empty string if it is specified that dbID doesn't have a cap style.

Return Type

STRING

Syntax

```
sch::dbGetLineCapStyle <dbId>
```

Parameters

Parameter	Type	Description
`dbID`	`DBID`	dbID of any This parameter is required.

Examples

```
sch::dbGetLineCapStyle db:00000017
```

Related Commands

dbGetLineJoinStyle

dbGetLineColor

It returns the line color of a drawing object, block shape, or block object. It returns #000000 for an invalid type of object.

Return Type

STRING

Syntax

```
sch::dbGetLineColor <dbId>
```

Parameters

Parameter	Type	Description
`dbID`	`DBID`	dbID of the drawing object or block shape or block object This parameter is required.

Examples

```
sch::dbGetLineColor db:00000034
#output:
#fcd054
```

Related Commands

dbGetLineCapStyle

dbGetLineJoinStyle

dbGetLineJoinStyle

It returns the join style of an object. Possible return values are bevel-join, miter-join, and round-join. It will return an empty string if it is specified that dbID doesn't have a join style.

Return Type

STRING

Syntax

```
sch::dbGetLineJoinStyle <dbId>
```

Parameters

Parameter	Type	Description
`dbId`	`DBID`	dbID of the object for which join style is required. This parameter is required.

Examples

```
sch::dbGetLineJoinStyle db:00000017
```

Related Commands

dbGetLineCapStyle

dbGetLineStyle

It returns the line style composing the item(s) (which can be a component, wire, etc.). The returned style can be of the following types:
- solid
- dash
- dot
- dash-dot
- dash-dot-dot
- multi-core
- single-core
- ribbon
- twisted

Return Type

STRING

Syntax

```
sch::dbGetLineStyle <itemId>
```

Parameters

Parameter	Type	Description
`itemId`	`STRING`	DBID of the Line This parameter is required.

Examples

```
#command to get the currently active page
set currentActivePage [sch::dbGetActivePage]
#command to get the currently selected items on the active page
set currentlySelectedItems [sch::dbGetSelectedItems $currentActivePage]
#loop over each item, get the line style for each, and print it
foreach item $currentlySelectedItems {set obtainedLineStyle [sch::dbGetLineStyle
$item];puts $obtainedLineStyle}
```

dbGetLineWidth

It returns the width of the line composing the item with id <itemId>.

Return Type

INT

Syntax

```
sch::dbGetLineWidth <itemId>
```

Parameters

Parameter	Type	Description
`itemId`	`STRING`	DBID of the line This parameter is required.

Examples

```
#command to get the current active page
set currentActivePage [sch::dbGetActivePage]
#command to get the currently selected items on the active page
set currentlySelectedItems [sch::dbGetSelectedItems $currentActivePage]
#loop over each item, get the line width for each, and print it.
foreach item $currentlySelectedItems { set obtainedLineWidth [sch::dbGetLineWidth
$item];puts $obtainedLineWidth }
```

dbGetLinkedItems

It returns the db identifiers (dbID) of the items which are connected with the selected object through a connecting link.

Return Type

LIST

Syntax

```
sch::dbGetLinkedItems <itemId>
```

Parameters

Parameter	Type	Description
`itemId`	`STRING`	DBID of the item This parameter is required.

Examples

```
#command to get the currently active page
set currentActivePage [sch::dbGetActivePage]
#command to get the list of dbIds for the currently selected items
set currentlySelectedItems [sch::dbGetSelectedItems $currentActivePage]
#command to get the item type for each linked item and print it
foreach item $currentlySelectedItems { set obtainedItemType [sch::dbGetItemType
$item];puts $obtainedItemType}
```

dbGetPage

It returns the ID of the page for the corresponding library, cell, and view.

Return Type

STRING

Syntax

```
sch::dbGetPage <library_name> <cell_name> <page_name>
```

Parameters

Parameter	Type	Description
`library_name`	`STRING`	Name of the library This parameter is required.
`cell_name`	`STRING`	Name of the cell This parameter is required.
`page_name`	`STRING`	Name of the page This parameter is required.

Examples

```
#command to store the library name using the following command in a variable called
libraryName
set libraryName [ cps::getDirectiveValue GLOBAL def_worklib ]
#command to store the cell name using the following command in a variable called
cellName
set cellName [ sch::dbGetCellNames $libraryName ]
#command to store the view name using following command in a variable called viewName
set viewName [ sch::dbGetPageNames $libraryName $cellName ]
sch::dbGetPage $libraryName $cellName $viewName
#for example: page(1)
```

dbGetPageItems

This command lists the DBIDs of all the items placed on the canvas.

Return Type

LIST

Syntax

```
sch::dbGetPageItems <dbID>
```

Parameters

Parameter	Type	Description
`dbID`	`DBID`	dbID of the page This parameter is required.

Examples

```
#command to store the dbID of the page in a variable called pageDbID and then get the
list
#of all the items placed on the canvas
set pageDbID [ sch::dbGetActivePage ]
sch::dbGetPageItems $pageDbID
#running the above command returns the following dbIds: db:00000029
#db:0000002a db:0000002b db:0000002c db:0000002d db:0000002e
#db:00000025 db:00000010.
#the active page contains one item placed on it with some visible properties.
#the command returns the dbIds of all the children along with the dbId of the parent
object.
```

Related Commands

dbGetActivePage

dbGetPageNames

This command lists the names of all the pages present in the design.

Return Type

LIST

Syntax

```
sch::dbGetPageNames <library_name> <cell_name>
```

Parameters

Parameter	Type	Description
`library_name`	`STRING`	Name of the library This parameter is required.
`cell_name`	`STRING`	Name of the cell This parameter is required.

Examples

```
#command to Store the library name using following command in a variable called
libraryName
set libraryName [ cps::getDirectiveValue GLOBAL def_worklib ]
#command to store the cell name using following command in a variable called cellName
set cellName [ sch::dbGetCellNames $libraryName ]
#command to list all the page names present in the design
sch::dbGetPageNames $libraryName $cellName
#For example: page(1) page(2)
```

Related Commands

dbGetPageItems

dbGetPageOfObject

Returns the page ID where the given DBID is placed.

Return Type

DBID

Syntax

```
sch::dbGetPageOfObject <dbid>
```

Parameters

Parameter	Type	Description
`dbid`	`DBID`	dbID of object for which page ID is required. This parameter is required.

Examples

```
set dbID [ sch::dbGetSelectedItems [ sch::dbGetActivePage ]]
sch::dbGetPageOfObject $dbid
#Output - db id of page db:0000001c
```

Related Commands

dbSelectObjectById

dbGetPageSizeType

This command returns the type of page whose ID is provided. The page types are: A, B, C, D, E, Custom, and Symbol.

Return Type

STRING

Syntax

```
sch::dbGetPageSizeType <pageId>
```

Parameters

Parameter	Type	Description
`pageId`	`DBID`	dbID of the page This parameter is required.

Examples

```
#store the dbId of the active page in the variable called activePageDbID
#and then show the size of the page
set activePageDbID [ sch::dbGetActivePage ]
sch::dbGetPageSizeType $activePageDbID
#for example: symbol
```

dbGetPagesOfComponent

It returns the list of all the pages on which the component exists in the design.

Return Type

LIST

Syntax

```
sch::dbGetPagesOfComponent <comp_spath>
```

Parameters

Parameter	Type	Description
`comp_spath`	`SPATH`	sPath of the selected component on the canvas This parameter is required.

Examples

```
#store the sPath of the component in the variable called compSpath.
#and then get the list of all the pages that contain the selected component
#in the current design
set compSpath [ sch::dbGetSPath [sch::dbGetSelectedItems [sch::dbGetActivePage]]]
sch::dbGetPagesOfComponent $compSpath
#for example: page(1)
```

Related Commands

dbGetPagesOfNet

dbGetPagesOfNet

This command lists the names of the pages on which a net exists with the provided sPath.

Return Type

LIST

Syntax

```
sch::dbGetPagesOfNet <net_spath>
```

Parameters

Parameter	Type	Description
`net_spath`	`SPATH`	sPath of the net on the canvas This parameter is required.

Examples

```
#command to store the sPath of the selected Net in the variable called netSPath
set netSPath [ sch::dbGetSPath [ sch::dbGetSelectedItems [ sch::dbGetActivePage ]]]
#command to returns the list of all the pages on which the net exists
sch::dbGetPagesOfNet $netSPath
#for example: page(1) page(2)
```

Related Commands

dbGetPagesOfComponent

dbGetPageSummary

It returns the summary of the page. The summary includes the name of the page and the count of blocks, connectors, graphic instances, groups, instances, net instances, NetGroups, nets, routes, route elements, and shapes present on the page. In case of failure, the function returns an empty list. The reason for failure can be due to an invalid or empty dbID of the page supplied as parameter.

Return Type

LIST

Syntax

```
sch::dbGetPageSummary <pageId>
```

Parameters

Parameter	Type	Description
`pageId`	`DBID`	dbID of the page This parameter is required.

Examples

```
#commands to return the summary of the currently active page
# get the pageId of the active page
set pgId [::sch::dbGetActivePage]
#get the page summary using the pageId
sch::dbGetPageSummary $pgId
#output: {name {Connectivity - Bus Tap S Pin(40)}} {num_blocks 0} {num_connectors 0}
#{num_graph_insts 1} {num_groups 1} {num_insts 3} {num_net_insts 2} {num_netgrous 2}
#{num_nets 8} {num_route_els 18} {num_routes 10} {num_shapes 13}
```

Related Commands

dbGetActivePage

dbGetPageNames

dbGetActivePageSpath

dbGetPageSizeType

dbGetPageItems

dbGetPage

dbGetParent

It returns the dbID of the parent of the object. In case of failure, it returns an empty string or an invalid dbID db:ffffffff.

Return Type

DBID

Syntax

```
sch::dbGetParent <itemId>
```

Parameters

Parameter	Type	Description
`itemId`	`DBID`	dbID of the object This parameter is required.

Examples

```
#command to return the parent's dbID of one of the selected objects on the active page
#get the pageId of the active page
set pgId [sch::dbGetActivePage]
#get the list of dbIDs of all the selected objects on the page
set selectedObjectIds [sch::dbGetSelectedItems $pgId]
#get the dbID of the first selected object in the list of dbIDs
set firstSelectedObjectId [lindex $selectedObjectIds 0]
#get the parent object's dbID of the first selected object
set parentObjectId [sch::dbGetParent $firstSelectedObjectId]

#Output: db:00000016
```

Related Commands

dbGetActivePage

dbGetSelectedItems

dbGetType

dbGetPinData

It returns the information of the pin. The returned information includes the pin name, its base name, pin type, pin direction, most significant bit, and least significant bit.

Return Type

LIST

Syntax

```
sch::dbGetPinData <pinId>
```

Parameters

Parameter	Type	Description
`pinId`	`DBID`	DBID of the pin of the component This parameter is required.

Examples

```
# after selecting a few pins on the page and running the following sequence of commands
returns the data of one of the pins
#get the pageId of the active page
set pgId [sch::dbGetActivePage]
#get the list of dbIDs of all the selected pins on the page
set selectedPinIds [sch::dbGetSelectedItems $pgId]
#get the dbID of the first selected pin in the list of pins' dbIDs
set firstSelectedPinId [lindex $selectedPinIds 0]
#get the data of the first selected pin
set pinData [sch::dbGetPinData $firstSelectedPinId]

#Output: {base a} {dir InOut} {lsb 0} {msb 0} {name A&lt;0&gt;} {type vector}
```

Related Commands

dbGetActivePage

dbGetSelectedItems

dbGetParent

dbGetPinNumber

dbGetBlockPins

dbIsPinRouteConnected

dbGetPinSide

Returns the information of the side of the system block pin with respect to the parent system block instance.
The expected output is TOP, BOTTOM, RIGHT, or LEFT.

Return Type

STRING

Syntax

```
sch::dbGetPinSide <pinId>
```

Parameters

Parameter	Type	Description
`pinId`	`DBID`	dbID of the system block pin currently selected on the canvas This parameter is required.

Examples

```
# after selecting a few system block pins on the page and running the following
sequence of commands returns the data of one of the pins
#get the pageId of the active page
set pgId [sch::dbGetActivePage]
#get the list of dbIDs of all the selected pins on the page
set selectedPinIds [sch::dbGetSelectedItems $pgId]
#get the dbID of the first selected pin in the list of pins' dbIDs
set firstSelectedPinId [lindex $selectedPinIds 0]
#get the side of the first selected pin
set pinSide [sch::dbGetPinSide $firstSelectedPinId]
```

dbGetPoints

It returns the following:
1. Starting points and the ending points of a line.
2. Coordinates of the top-left and the bottom-right corners of the rectangle.
3. Coordinates for the top-left and bottom-right corners of the bounding rectangle of the ellipse or block shape.
4. All the points of a connector shape.

In case of failure, the function returns an empty list.

Return Type

LIST

Syntax

```
sch::dbGetPoints <shape>
```

Parameters

Parameter	Type	Description
`shape`	`DBID`	dbID of the shape This parameter is required.

Examples

```
after selecting a line on the page, the following commands get the points of the line-
#get the pageId of the active page
set pgId [sch::dbGetActivePage]
#get the dbID of the selected line
set selectedLineId [sch::dbGetSelectedItems $pgId]
#get the points of the line
set linePts [sch::dbGetPoints $selectedLineId]
#output: {1930400 787400} {2232660 779780}
#after selecting an ellipse on the page, the following commands get the top left and
#bottom right points of the bounding rectangle of the ellipse
#get the pageId of the active page
set pgId [sch::dbGetActivePage]
#get the dbID of the selected ellipse
set selectedEllipseId [sch::dbGetSelectedItems $pgId]
#get the points of the ellipse
set ellipsePts [sch::dbGetPoints $selectedEllipseId]
#output: {1930400 787400} {2232660 779780}
#after selecting a connector on the page, the following commands
#get the points of the line
#get the pageId of the active page
set pgId [sch::dbGetActivePage]
#get the dbID of the selected connector
set selectedConnectorId [sch::dbGetSelectedItems $pgId]
#get the points of the ellipse
set connectorPts [sch::dbGetPoints $selectedConnectorId]
#output: {1244600 1168400} {1460500 1168400} {1460500 1397000}
```

Related Commands

dbGetActivePage

dbGetSelectedItems

dbGetType

dbGetShapeBBox

dbGetPos

It returns the list containing the center X-Y coordinates of the object. In case of failure, the command returns a list with the position {0 0}.

Return Type

LIST

Syntax

```
sch::dbGetPos <shape>
```

Parameters

Parameter	Type	Description
`shape`	`STRING`	dbID of the shape object This parameter is required.

Examples

```
#after selecting a component instance on the page, the following commands get the
center XY co-ordinate of component-
#get the pageId of the active page
set pgId [sch::dbGetActivePage]
#get the dbID of the selected object
set selectedObjId [sch::dbGetSelectedItems $pgId]
#get the position, i.e. the center XY co-ordinate of the object
set objPos [sch::dbGetPos $selectedObjId]
#output: 1638300 1003300
```

Related Commands

dbGetActivePage

dbGetSelectedItems

dbGetPoints

dbGetHotSpot

dbGetProperties

Returns a list of lists containing information about properties on the object identified by item-Id in the specified occurrence path. Each list has the following information:
{<property name> <property value> <is name visible> <is value visible> <property value type> <reserved for future use> <reserved for future use>}

In this example, {VOLTAGE 6.3V 0 1 String 0 0}, the property VOLTAGE has a value of 6.3V with the value displayed on the schematic.

Return Type

LIST

Syntax

```
sch::dbGetProperties <itemId> <occSPath>
```

Parameters

Parameter	Type	Description
`itemId`	`DBID`	dbID of a component or item This parameter is required.
`occSPath`	`STRING`	Occurrence sPath of the item This parameter is required.

Examples

```
set props [sch::dbGetProperties [sch::dbGetSelectedItems [sch::dbGetActivePage]]
[sch::dbGetSPathForActiveTab]]
# returns a list of property objects
puts $props
{LOCATION C79 0 1 String 0 0} {PACK_TYPE 0603 0 0 String 0 0} {PART_NAME CAP 0 0 String
0 0} {PART_NUMBER CDN-CAP-0007 0 0 String 0 0} {SEC 1 0 0 String 0 0} {TOLERANCE 40% 0
1 String 0 0} {VALUE 22uF 0 1 String 0 0} {VOLTAGE 6.3V 0 1 String 0 0}

# The number of properties is equal to the length of the list
puts "Number of properties: [llength $props]"
```

Related Commands

dbGetSelectedItems

dbGetActivePage

dbGetPropNameVal

It returns the name and the value pair of the property of a component or item.

Return Type

DBStrMap

Syntax

```
sch::dbGetPropNameVal <item_id> <spath_of_design>
```

Parameters

Parameter	Type	Description
`item_prop_id`	`DBID`	dbID of a component or item This parameter is required.
`spath_of_design`	`STRING`	Occurrence sPath of the item This parameter is required.

Examples

```
#command to find the dbID of the selected item on the canvas
sch::dbGetSelectedItems [ sch::dbGetActivePage ]
#this returns the dbID, such as db:0000001c
sch::dbGetPropNameVal db:0000001c @worklib.trial10(tbl_1)
#running the command returns {IMPLEMENTATION 2n7000}.)
#running the command returns {IMPLEMENTATION 2n7000}.
#returns the name along with the value of the property.
#The first element is the name and the second element is the value of the property.
```

Related Commands

dbGetSelectedItems

dbGetActivePage

dbGetPropVisibility

It returns the visibility of a property name-value pair. The following table illustrates the values returned and their meanings:
0: The name and value of the property are not visible
1: The name of the property is visible, the value is not visible
2: The name of the property is not visible, the value is visible
3: The name and value of the property are visible

Return Type

INTEGER

Syntax

```
sch::dbGetPropVisibility <itemId>
```

Parameters

Parameter	Type	Description
`itemId`	`DBID`	dbID of a component or item This parameter is required.

Examples

```
#finds the dbID of the selected item on the canvas
set id [sch::dbGetSelectedItems [ sch::dbGetActivePage ]]
#get the visibility of the property name-value pair
sch::dbGetPropVisibility $id
```

Related Commands

dbGetSelectedItems

dbGetActivePage

dbGetRootDesignName

It returns the name of the top-level/root design currently being worked upon.

Return Type

STRING

Syntax

```
sch::dbGetRootDesignName
```

Examples

```
#consider that the design at the location "/home/user_name/testdsn/workdesign.cpm"
#is currently open
#the following command would then return "workdesign"
sch::dbGetRootDesignName
```

dbGetRotationValueOfProp

It returns the rotation value of a property relative to its parent item. This means that if an item and its property are both oriented at 90 degrees, this routine will return 0 or 360.

Return Type

DOUBLE

Syntax

```
sch::dbGetRotationValueOfProp <dbId>
```

Parameters

Parameter	Type	Description
`dbId`	`STRING`	dbID of the property This parameter is required.

Examples

```
#get the current active page
set currentActivePage [sch::dbGetActivePage]
#get the current selected properties
set currentlySelectedItem [sch::dbGetSelectedItems $currentActivePage]
#loop over the dbIds for the currently selected items, get the fill color,
#and print it out
foreach item $currentlySelectedItem { set rotationValue [sch::dbGetRotationValueOfProp
$item]; puts "Rotation for item $item is $rotationValue "}
```

Related Commands

dbGetRotationValue

dbGetSegments

Returns a list of segment objects of a route. Each segment object is a list of the starting point, the ending point, and the segment type. Supported segment types are wire-segment, connect line, bus-segment, and junction. The Tcl values corresponding to these types are sch::DBTWireSegment, sch::DBTConnectLine, sch::DBTBusSegment and sch::DBTJunction respectively.

Returns an empty list on failure.

Return Type

LIST

Syntax

```
sch::dbGetSegments <routeId>
```

Parameters

Parameter	Type	Description
`routeId`	`DBID`	dbID of a route This parameter is required.

Examples

```
#Print all junctions of the selected route
set routeId [::sch::dbGetSelectedItems [::sch::dbGetActivePage]]
set segments [sch::dbGetSegments $routeId]
foreach segment $segments {
set segmentStart [lindex $segment 0]
set segmentEnd [lindex $segment 1]
set segmentType [lindex $segment 2]
if {$::sch::DBTJunction == $segmentType} {
puts "Found junction - $segmentStart, $segmentEnd"
}
}
```

Related Commands

dbGetActivePage

dbGetSelectedItems

dbGetSelectedItems

It returns a list of dbIDs of the items selected on a page. In case of a failure or no object selected, it returns an empty list.

Return Type

LIST

Syntax

```
sch::dbGetSelectedItems <pageId>
```

Parameters

Parameter	Type	Description
`pageId`	`DBID`	dbID of the page This parameter is required.

Examples

```
#after selecting a set of objects on the page, running he following commands
#get the list of dbIDs of the selected objects
#get the pageId of the active page
set pgId [sch::dbGetActivePage]
#get the list dbIDs of the selected objects
set selectedObjectIds [sch::dbGetSelectedItems $pgId]
#output: db:0000006f db:00000074 db:00000058 db:00000071 db:00000070
#db:00000019 db:00000075 db:00000076 db:00000077 db:0000001f db:00000021
#db:00000072 db:00000022 db:00000027 db:00000017
```

Related Commands

dbGetActivePage

dbGetType

dbGetSelectedItemsEx

Command returns a list of dbIDs of the items selected on a page.
If an object and its child object both are selected (e.g. component and its property) then shouldIgnoreChildren specifies whether both objects should be returned or only parent object is required.
Value $::sch::DBFalse will not ignore child objects and will return all objects including children.

Return Type

LIST

Syntax

```
sch::dbGetSelectedItems <pageId> <shouldIgnoreChildren>
```

Parameters

Parameter	Type	Description
`pageId`	`DBID`	dbID of the page This parameter is required.
`shouldIgnoreChildren`	`DBBool`	Flag tp specify whether child objects along with parent object should be returned This parameter is required.

Examples

```
#after selecting a component and its property/pin on the page, running he following
commands

#get the pageId of the active page
set pgId [sch::dbGetActivePage]

#get the list dbIDs of the selected objects
sch::dbGetSelectedItemsEx [ sch::dbGetActivePage ] $::sch::DBTrue
db:0000001e
Tcl&gt; sch::dbGetSelectedItemsEx [ sch::dbGetActivePage ] $::sch::DBFalse
db:0000001e db:0000001f
```

Related Commands

dbGetSelectedItems

dbGetActivePage

dbGetShapeBBox

It returns a list containing the top-left and the bottom-right points of the bounding box of the component, pin, or the graphical shape. Bounding box of a component is the enclosing rectangle of the component without the pins.

In case of a failure, it returns an empty list.

Return Type

BBOX

Syntax

```
sch::dbGetShapeBBox <shape>
```

Parameters

Parameter	Type	Description
`shape`	`DBID`	dbID of a component, pin, or graphical shape This parameter is required.

Examples

```
#After selecting a component on the page, running the following commands
#get the bounding box of the component excluding its pins
#get the pageId of the active page
set pgId [sch::dbGetActivePage]
#get the dbID of the selected component
set selectedComponentId [sch::dbGetSelectedItems $pgId]
#get the bounding box of the component excluding pins
set componentBBox [sch::dbGetShapeBBox $selectedComponentId]
#output: {2021840 909320} {2042160 970280}
```

Related Commands

dbGetActivePage

dbGetSelectedItems

dbGetType

dbGetPoints

dbGetSPath

It returns the sPath of the specified item, if available.

Return Type

STRING

Syntax

```
sch::dbGetSPath <item_id>
```

Parameters

Parameter	Type	Description
`item_id`	`STRING`	ID of the item This parameter is required.

Examples

```
sch::dbGetSPath db:00000046
#running the above command returns @worklib.trial17(tbl_1):\N7\
#here, the command is passed the dbId of the instance placed on the active page.
#the returned value is the sPath of the instance.
```

dbGetSPathForActiveTab

Returns the in context sPath of the active tab for the schematic. This command can be used with other commands to extract connectivity-related information.

Return Type

STRING

Syntax

```
sch::dbGetSPathForActiveTab
```

Examples

```
sch::dbGetSPathForActiveTab
#running the above command returns @worklib.trial17(tbl_1)
#here, the returned value is the sPath of the currently active tab
#this path can be used to get properties of object placed on the page
sch::dbGetProperties [ getSel ] [ sch::dbGetSPathForActiveTab ]
#Above command will return in-context properties of selected object on active tab.
```

Related Commands

dbGetProperties

dbGetPropNameVal

dbGetStyleInformation

It returns the style information of an item.

Return Type

STRING

Syntax

```
sch::dbGetStyleInformation <dbId>
```

Parameters

Parameter	Type	Description
`dbId`	`STRING`	dbID of the item This parameter is required.

Examples

```
#in this example, the style information of the currently selected object
#in the schematic editor is returned.
sch::dbGetStyleInformation [sch::dbGetSelectedItems [sch::dbGetActivePage]]
#output:
#{alt-fill-color #000000} {bold false} {custom-image-minimum-line-width 6}
#{fill-color #ffffff} {fill-style none} {font-color #000000} {font-name Arial} {font-
size 5}
#{inst-prop-font-size 5} {italic false} {item-opacity 1} {line-cap-style round-join}
#{line-color #4200ff} {line-join-style round-cap} {line-style solid} {line-width 1}
#{pin-prop-font-size 4} {text-margin 0} {text-wordwrap false} {underline false}
```

dbGetStyleName

It returns the style name of an item.

Return Type

STRING

Syntax

```
sch::dbGetStyleName <dbId>
```

Parameters

Parameter	Type	Description
`dbId`	`STRING`	dbID of the Text This parameter is required.

Examples

```
sch::dbGetStyleName db:00000015
```

dbGetTableColCount

It returns the number of columns present in a table.

Return Type

INT

Syntax

```
sch::dbGetTableColCount <table_id>
```

Parameters

Parameter	Type	Description
`table_id`	`STRING`	dbID of the table This parameter is required.

Examples

```
sch::dbGetTableColCount db:00000015
sch::dbGetTableColCount [sch::dbGetSelectedItems [sch::dbGetActivePage]]
#this command returns the total number of columns in a table
```

dbGetTablePlainText

It returns the text in a cell in plaintext format, without html tags

Return Type

STRING

Syntax

```
sch::dbGetTablePlainText <table_id> <row_num> <column_num>
```

Parameters

Parameter	Type	Description
`table_id`	`STRING`	The dbID of the table. This parameter is required.
`row_num`	`INT`	The row number. This parameter is required.
`column_num`	`INT`	The column number. This parameter is required.

Examples

```
sch::dbGetTablePlainText db:00000015 1 0
```

dbGetTableRichText

It returns the rich text from a cell.

Return Type

STRING

Syntax

```
sch::dbGetTableRichText <table_id> <row_num> <column_num>
```

Parameters

Parameter	Type	Description
`table_id`	`STRING`	The dbID of the table. This parameter is required.
`row_num`	`INT`	The row number. This parameter is required.
`column_num`	`INT`	The column number. This parameter is required.

Examples

```
sch::dbGetTableRichText db:00000015 1 0
```

dbGetTableRowCount

It returns the total number of rows in a table.

Return Type

INT

Syntax

```
sch::dbGetTableRowCount <table_id>
```

Parameters

Parameter	Type	Description
`table_id`	`STRING`	dbID of the table This parameter is required.

Examples

```
sch::dbGetTableRowCount db:00000015
```

dbGetType

Gets the type of the item.

Return Type

INT

Syntax

```
sch::dbGetType <item_id>
```

Parameters

Parameter	Type	Description
`item_id`	`STRING`	the dbID of an item This parameter is required.

Examples

```
sch::dbGetType db:0000001c
#this command returns an integer value which is
#the type of the item passed.
```

dbIsByPassItem

Returns whether the item is a part of the bypass object or not.

Return Type

NONE

Syntax

```
sch::dbIsByPassItem <item_id>
```

Parameters

Parameter	Type	Description
`item_id`	`STRING`	the dbID of an item This parameter is required.

dbIsFontBold

It returns whether the font is bold or not. The value returned is 1 for yes and 0 for no.

Return Type

BOOLEAN

Syntax

```
sch::dbIsFontBold <dbId>
```

Parameters

Parameter	Type	Description
`dbId`	`STRING`	dbID of the text This parameter is required.

Examples

```
sch::dbIsFontBold db:00000015
```

dbIsFontItalic

Use dbIsFontItalic to check if the font of the text item is italicized or not. It returns 1 if the font is in italics, otherwise 0.

Return Type

BOOL

Syntax

```
sch::dbIsFontItalic <dbId>
```

Parameters

Parameter	Type	Description
`dbId`	`STRING`	dbID of the text This parameter is required.

Examples

```
sch::dbIsFontItalic db:00000015
#the command will return 1 if the font is italic otherwise 0.
```

dbIsFontUnderlined

Use dbIsFontUnderlined to check if the text is underlined or not. It returns 1 for yes and 0 for no.

Return Type

BOOLEAN

Syntax

```
sch::dbIsFontUnderlined <dbId>
```

Parameters

Parameter	Type	Description
`dbId`	`STRING`	dbID of the Text This parameter is required.

Examples

```
sch::dbIsFontUnderlined db:00000015
```

dbIsMasterOccProp

This command is used to check whether this property is present on the master design.

Return Type

INT

Syntax

```
sch::dbIsMasterOccProp <prop_id>
```

Parameters

Parameter	Type	Description
`prop_id`	`STRING`	property ID This parameter is required.

Examples

```
sch::dbIsMasterOccProp db:00000017
```

dbIsPinRouteConnected

It identifies whether the specific pin is connected to a specific route. It returns 1 if the pin is connected to the route and 0 if the pin is not connected to the route.

Return Type

BOOL

Syntax

```
sch::dbIsPinRouteConnected <pin_id> <route_id>
```

Parameters

Parameter	Type	Description
`pin_id`	`DBID`	dbID of the pin. This parameter is required.
`route_id`	`DBID`	dbID of the route. This parameter is required.

Examples

```
#first, the DbID of the current page is used to find the DbID of the selected pin.
sch::dbGetSelectedItems [ sch::dbGetActivePage ]
#this gives a pin's DbID, such as 'db:0000001c'
#next, the DbID of a selected route:
sch::dbGetSelectedItems [ sch::dbGetActivePage ]
#this gives the route's ID, such as db:0000001d
#now, let's check if this pin is connected to the route:
sch::dbIsPinRouteConnected db:0000001c db:0000001d
#this returns '1' as this pin is connected to the route.
#the next example shows the result in case of an unconnected pin.
sch::dbIsPinRouteConnected db:0000001c db:0000001d
#this returns '0' as this pin is not connected to the route.
```

Related Commands

dbGetSelectedItems

dbGetActivePage

dbIsValid

Use dbIsValid to check if the specified dbID is in use by any object or component in the currently open design. It returns 0 if the given dbID is not used by a component or item in the current design, and returns 1 if a component is using the given dbID in the design.

Return Type

BOOLEAN

Syntax

```
sch::dbIsValid <db_id>
```

Parameters

Parameter	Type	Description
`db_id`	`DBID`	dbID of a component or item This parameter is required.

Examples

```
#let's find the dbID of the selected item on the canvas
sch::dbGetSelectedItems [ sch::dbGetActivePage ]
#this returns the dbID, such as db:0000001c
#now, check if this dbID is in use
sch::dbIsValid db:0000001c
#as this component is in the design, the return value is '1' this means it is a valid
dbID.
#now, check another dbID
sch::dbIsValid db:0000001d
#this returns a '0' meaning this is an invalid dbID for this design, as it does not
exist in the design.
```

dbIsZeroSegmentNet

Use dbIsZeroSegmentNet to check if the net has zero segments or not.

Return Type

BOOLEAN

Syntax

```
sch::dbIsZeroSegmentNet <netSpath>
```

Parameters

Parameter	Type	Description
`netSpath`	`STRING`	sPath of net This parameter is required.

Examples

```
#this command extracts the sPath of the currently selected net:
sch::dbIsZeroSegmentNet [ sch::dbGetSPath [ sch::dbGetSelectedItems [
sch::dbGetActivePage ] ] ]
#as this is an existing net, it is bound to have segments, the return value will be '0'
```

dbName2Type

It returns System Capture's internal type numbers for the specified component, such as NetGroup, bus, route, arc, connector, and so on. It returns -1 for the unknown component name.

Return Type

DBTYPE

Syntax

```
sch::dbName2Type <item_name>
```

Parameters

Parameter	Type	Description
`item_name`	`STRING`	Name of the component This parameter is required.

Examples

```
#command to find the type number of object types:
sch::dbName2Type Rect
#output: 8
sch::dbName2Type Inst
#output: 16
sch::dbName2Type Power
#output: 21
sch::dbName2Type abcd
#output: -1
```

dbParseSignal

It returns the list containing the details of a signal or net. This includes base name, lsb value, msb value, and the type of the net.

Return Type

DBSRRMAP

Syntax

```
sch::dbParseSignal <signalName>
```

Parameters

Parameter	Type	Description
`signalName`	`STRING`	name of the signal This parameter is required.

Examples

```
#if you run this command with a signal, 'signal1'
sch::dbParseSignal signal1
#the following values are shown:
#{base signal1} {lsb -1} {msb -1} {type scalar}
```

dbParseSPath

Returns a parsed map as key-value strings pairs for a given sPath.

Return Type

Map

Syntax

```
sch::dbParseSPath <sPath_string>
```

Parameters

Parameter	Type	Description
`sPath_string`	`STRING`	The sPath of the object This parameter is required.

Examples

```
#get the pageId of the active page
set pgId [sch::dbGetActivePage]
#get the list of dbIDs of all the selected Nets on the page
set selectedNetIds [sch::dbGetSelectedItems $pgId]
#get the dbID of the first selected Net in the list of Nets' dbIDs
set firstSelectedNetId [lindex $selectedNetIds 0]
#get the SPath of the first selected Net
set netSPath [sch::dbgetSPath $firstSelectedNetId]

#Output - @worklib.root_design(tbl_1):\N1\(15:0)

#Parse the spath
sch::dbParseSPath $netSPath

#Output - {BlockName root_design} {connID N1} {lowerBit 15} {upperBit 0}
```

dbSelectObjectById

Selects an object using its DBID on a page.

If operations need to be performed on the page where the selected object is then set shouldOpenTab to value $::sch::DBTrue
However, if the operation is independent of the active page or you do not want to change the active page then set $::sch::DBFalse.
All editor write commands(modifying objects on design), should pass shouldOpenTab to value $::sch::DBTrue

Return Type

BOOL

Syntax

```
sch::dbSelectObjectById <dbId> <shouldOpenTab>
```

Parameters

Parameter	Type	Description
`dbId`	`DBID`	DBID of schematic page object to be selected This parameter is required.
`shouldOpenTab`	`DBBool`	$::sch::DBTrue - then the page containing the object is opened if not already active. $::sch::DBFalse – page will not switch This parameter is required.

Examples

```
set pageDbID [ sch::dbGetActivePage ]
set objs [sch::dbGetPageItems $pageDbID ]
#Returns the dbIds of all objects on a page. For example, db:0000001d db:0000001e
db:0000001f db:00000020 db:00000021
set dbid [ lindex $objs 0 ]
sch::dbSelectObjectById $dbid $::sch::DBTrue
#command selects object db:0000001d
delete
#As delete is edit/write command(modifying objects on design), shouldOpenTab value
$::sch::DBTrue was mentioned in sch::dbSelectObjectById.
```

Related Commands

dbSelectRouteSegmentsById

dbUnselectObjectById

dbSelectObjectByIdEx

Used to select an object using DBID of any object on a page. This command is extended version of sch::dbSelectObjectById command with addition parameter to accept auto zoom option on selection.

Return Type

BOOL

Syntax

```
sch::dbSelectObjectByIdEx <dbId> <shouldOpenTab> <shouldAutoZoom>
```

Parameters

Parameter	Type	Description
`dbId`	`DBID`	DBID of schematic page object to be selected This parameter is required.
`shouldOpenTab`	`DBBool`	$::sch::DBTrue - then the page containing the object will be opened if not already active. $::sch::DBFalse – page will not switch This parameter is required.
`shouldAutoZoom`	`DBBool`	$::sch::DBTrue - selected object will be zoomed in. $::sch::DBFalse – selection will not be zoomed This parameter is required.

Examples

```
set pageDbID [ sch::dbGetActivePage ]
set objs [sch::dbGetPageItems $pageDbID ]
#running the above command returns the dbIds of all objects on a page. For example,
db:0000001d db:0000001e db:0000001f db:00000020 db:00000021
set dbid [ lindex $objs 0 ]
sch::dbSelectObjectByIdEx $dbid $::sch::DBTrue $::sch::DBFalse
#command will select object db:0000001d and would not zoom to selected object because
last parameter in commands is provided with value false.
```

dbSelectRouteSegmentsById

Selects the segments of the route for the given dbID.

Return Type

BOOL

Syntax

```
sch::dbSelectRouteSegmentsById <dbId> <segmentsList> <shouldOpenTab>
```

Parameters

Parameter	Type	Description
`dbId`	`DBID`	dbID of route to be selected This parameter is required.
`segmentsList`	`DBSegmentList`	list of segments to be selected This parameter is required.
`shouldOpenTab`	`DBBool`	$::sch::DBTrue - then the page containing the object will be opened if not already active $::sch::DBFalse – page will not switch This parameter is required.

Examples

```
# Select a wire/bus on the schematic page and execute these commands
set dbidroute [ sch::dbGetSelectedItems [ sch::dbGetActivePage ]]
sch::dbGetSegments $dbidroute
#Above command will list of all segments of route.
# {{1701800 1219200} {1701800 2146300} 28} {{1701800 2146300} {2108200 2146300} 28}
{{3556000 2146300} {3962400 2146300} 28} {{3962400 1219200} # {3962400 2146300} 28}

sch::dbSelectRouteSegmentsById $dbidroute {{{1701800 1219200} {1701800 2146300} 28}
{{3556000 2146300} {3962400 2146300} 28} } $::sch::DBTrue
#This will select provided segments of route.

In the above example, the following are the segment type identifiers:
28 - Wire Segment, Variable name - sch::DBTWireSegment
29 - Rats/Connect line, Variable name - sch::DBTConnectLine
30 - Bus Segment, Variable name - sch::DBTBusSegment
```

Related Commands

dbUnselectRouteSegmentsById

dbSelectObjectById

dbUnselectObjectById

dbSelectRouteSegmentsByIdEx

Selects the segments of the route for the given dbID. This command is extended version of sch::dbSelectRouteSegmentsById command with addition parameter to accept auto zoom option on selection.

Return Type

BOOL

Syntax

```
sch::dbSelectRouteSegmentsByIdEx <dbId> <segmentsList> <shouldOpenTab> <shouldAutoZoom>
```

Parameters

Parameter	Type	Description
`dbId`	`DBID`	dbID of route to be selected This parameter is required.
`segmentsList`	`DBSegmentList`	list of segments to be selected This parameter is required.
`shouldOpenTab`	`DBBool`	$::sch::DBTrue - then the page containing the object will be opened if not already active $::sch::DBFalse – page will not switch This parameter is required.

Examples

```
# Select a wire/bus on the schematic page and execute these commands
set dbidroute [ sch::dbGetSelectedItems [ sch::dbGetActivePage ]]
sch::dbGetSegments $dbidroute
#Above command will list of all segments of route.
# {{1701800 1219200} {1701800 2146300} 28} {{1701800 2146300} {2108200 2146300} 28}
{{3556000 2146300} {3962400 2146300} 28} {{3962400 1219200} # {3962400 2146300} 28}

sch::dbSelectRouteSegmentsById $dbidroute {{{1701800 1219200} {1701800 2146300} 28}
{{3556000 2146300} {3962400 2146300} 28} } $::sch::DBTrue $::sch::DBFalse
#This will select provided segments of route and would not zoom to selected object
because last parameter in commands is provided with value false.

In the above example, the following are the segment type identifiers:
28 - Wire Segment, Variable name - sch::DBTWireSegment
29 - Rats/Connect line, Variable name - sch::DBTConnectLine
30 - Bus Segment, Variable name - sch::DBTBusSegment
```

dbType2Name

It returns the object name of a given component's type. It returns 'UNKNOWN' for an unrecognized type.

Return Type

STRING

Syntax

```
sch::dbType2Name <type>
```

Parameters

Parameter	Type	Description
`type`	`INT`	Type of the component This parameter is required.

Examples

```
#command to find the type name of the selected item on the canvas:
sch::dbType2Name [ sch::dbGetType [ sch::dbGetSelectedItems [ sch::dbGetActivePage ] ]
]
#this would return 'Block' as a block was selected.
#to find the name of the objects with type '8'
sch::dbType2Name 8
#this returns 'Rect' which is the object name.
```

Related Commands

dbGetType

dbGetSelectedItems

dbGetActivePage

dbUnselectObjectById

Unselects the object whose dbId is given.

Return Type

BOOL

Syntax

```
sch::dbUnselectObjectById <dbid> <shouldOpenTab>
```

Parameters

Parameter	Type	Description
`dbid`	`DBID`	dbID of object to be unselected This parameter is required.
`shouldOpenTab`	`DBBool`	$::sch::DBTrue - then the page containing the object will be opened if not already active $::sch::DBFalse – page will not switch This parameter is required.

Examples

```
set dbid [ sch::dbGetSelectedItems [ sch::dbGetActivePage ]]
sch::dbUnselectObjectById $dbid $::sch::DBTrue
```

Related Commands

dbSelectObjectById

dbSelectRouteSegmentsById

dbUnselectRouteSegmentsById

dbUnselectRouteSegmentsById

Unselects the segments of the route with the given dbID.

Return Type

BOOL

Syntax

```
sch::dbUnselectRouteSegmentsById <dbId> <segments> <shouldOpenTab>
```

Parameters

Parameter	Type	Description
`dbId`	`DBID`	dbID of the route This parameter is required.
`segments`	`DBSegmentList`	list of segments to be unselected This parameter is required.
`shouldOpenTab`	`DBBool`	$::sch::DBTrue - then the page containing the object will be opened if not already active $::sch::DBFalse – page will not switch This parameter is required.

Examples

```
set dbidroute [ sch::dbGetSelectedItems [ sch::dbGetActivePage ]]
sch::dbGetSegments $dbidroute
#Above command will list of all segments of route.
# {{1701800 1219200} {1701800 2146300} 28} {{1701800 2146300} {2108200 2146300} 28}
{{3556000 2146300} {3962400 2146300} 28} {{3962400 1219200} # {3962400 2146300} 28}

sch::dbSelectRouteSegmentsById $dbidroute {{{1701800 1219200} {1701800 2146300} 28}
{{3556000 2146300} {3962400 2146300} 28} } $::sch::DBTrue
#This will select provided segments of route.

sch::dbUnselectRouteSegmentsById $dbidroute {{{1701800 1219200} {1701800 2146300} 28}
{{3556000 2146300} {3962400 2146300} 28} } $::sch::DBTrue
#This will unselect provided segments of route.

In the above example, the following are the segment type identifiers:
28 - Wire Segment, Variable name - sch::DBTWireSegment
29 - Rats/Connect line, Variable name - sch::DBTConnectLine
30 - Bus Segment, Variable name - sch::DBTBusSegment
```

Related Commands

dbSelectRouteSegmentsById

dbSelectObjectById

dbUnselectObjectById

getBlockPageNumber

It returns list of page numbers for a block. Block name is given as argument. Page numbers of the block are for block in master mode.

Return Type

List

Syntax

```
schPageUtils::getBlockPageNumber blockName
```

Examples

```
package require schPageUtils

set x [schPageUtils::getDesignPagesAllInfo]
set size [schPageUtils::getPagesInfoSize $x]
for {set i 1} {$i &lt;= $size} {incr i} {

puts "--------------------"
puts "Page : [schPageUtils::getPageName $x $i]"
puts "Id : [schPageUtils::getPageId $x $i]"
puts "TOC status : [schPageUtils::isPageTOC $x $i]"
puts "Print status : [schPageUtils::isPagePrintable $x $i]"
}
```

getDesignPagesAllInfo

Returns information ‘pageInfo’ for all the pages in the design. This requires package schPageUtils to be included

Return Type

pageInfo

Syntax

```
schPageUtils::getDesignPagesAllInfo
```

Examples

```
package require schPageUtils
schPageUtils::getDesignPagesAllInfo
```

getIndexList

It returns list of page indexes for which property name of page and its value matched with the property name and value given as input. It returns list of indexes having property name propName and its value propVal.

Return Type

List

Syntax

```
schPageUtils::getIndexList propName propVal
```

getPageInfoSize

It takes ‘pageInfo’ as argument. It returns the total number of pages in the design. pageInfo returned from getDesignPagesAllInfo command is used as input.

Return Type

Integer

Syntax

```
schPageUtils::getPagesInfoSize pageInfo
```

getPageName

It takes ‘pageInfo’ and index into ‘pageInfo’ as argument. It returns name of the page at index given as argument.
pageInfo returned from getDesignPagesAllInfo command is used as input.
Range of index is from 1 to total number of pages in design.

Return Type

String`

Syntax

```
schPageUtils::getPageName pageInfo index
```

getPrintablePageNumbers

It returns list of page numbers which are printable.

Return Type

List

Syntax

```
schPageUtils::getPrintablePageNumbers
```

getSelectionFilterChecked

Returns items in the selection filter which are selected.

Return Type

List

Syntax

```
schSelectionFilterUtils::getSelectionFilterChecked
```

Examples

```
{Block Shape} Bus Component {Connect Line} {Drawing Object} NetGroup Note Pin Property
{Signal Name} {Special Symbols} {System Design Objects} Table Wire
```

getSelectionFilterUnchecked

It returns items in the selection filter which are not selected.

Return Type

List

Syntax

```
schSelectionFilterUtils::getSelectionFilterUnchecked
```

Examples

```
{All Objects} {Block Arrow}
```

getSpathListOfComponent

This command takes a refdes Id of the component and the block name where the symbol is placed and returns a list of spath for the refdes and block name entered.
This SPath from this command can be used as input to commands such as sch::dbGetPagesOfComponent , sch::dbGetInfoFromInstanceSpath

Return Type

List

Syntax

```
sdaConn::getSpathListOfComponent <refdes> <blockName>
```

Parameters

Parameter	Type	Description
`refdes`	`STRING`	Reference Designator of an instance whose SPaths need to be fetched. This parameter is required.
`blockName`	`STRING`	Specify the block name in which component is placed. This parameter is required.

Examples

```
# In this example, we have a symbol placed multiple times on a schematic page of block
named ref_5g, with a refdes number.
# Refdes number is available on right hand property pane after clicking the symbol.
sdaConn::getSpathListOfComponent D15 ref_5g
# This will return a list of comma separated SPath(s) of the selected components whose
refdes id has been passed.
# Output - @worklib.ref_5g(tbl_1):\\I791\\
```

Related Commands

dbGetPagesOfComponent

dbGetInfoFromInstanceSpath

isPageTOC

It tells whether a page in System capture design is TOC or not.
pageInfo returned from getDesignPagesAllInfo command is used as input. Range of index is from 1 to total number of pages in design.
It returns true if page is TOC, false otherwise

Return Type

String

Syntax

```
schPageUtils::isPageTOC pageInfo index
```

isValidBlockName

Checks whether a block is valid or not
Returns 1 if 'Yes', else 'None'

Return Type

BOOL

Syntax

```
isValidBlockName <blockName>
```

Parameters

Parameter	Type	Description
`blockName`	`STRING`	name of the block to be checked This parameter is required.

Examples

```
isValidBlockName new_block
```

4

Logical View

The logical view gives the hierarchical view of the schematic. That is, blocks available in the design, instantiations of these blocks, and logical connectivity of the blocks and instances. This section covers the APIs available for logical traversal of a System Capture design. Logical view and physical view commands enable schematic designers and engineers working with System Capture to retrieve physical and logical information about the design through supported TCL APIs. Commands can also be used to read the information about the design and its objects, such as component instances and nets, loaded on the connectivity server. Modifications to the design objects or their properties are not supported. A design can enumerate the following:

- Instances in the design
- Nets in the design
- Aliases in the design
- Connections in the design
- Cell interfaces used in the design

Constituents of logical view are:

- **ICellInterface Object**

A cell interface object represents the definition of the interfaces for a cell. This includes the library, cell, and view this cell is loaded from. In addition, the interface ports of the cell and the list of parameters this cell can have are also included in the interface definition.

- **IDesign Object**

The IDesign class represents the hierarchical block. The interface of a design object is represented by its cell interface, which contains the interface port specification, library, cell, and view information.

- **IInstance Object**

An instance in a design is represented using an IInstance object. IInstances are instantiations of a particular cell interface object. In addition, instances can contain values for parameters specified with the cell interface. Finally, instances enumerate the instance terminals on themselves.

- **INet Object**

A net in design is represented using an INet object. Nets have the following attributes:

1. Name
2. Net Width
3. Scope (local, global, or interface)

In addition, nets can enumerate the connections they make.

- **ITerm Object**

Terminals represent ports on a cell. Terminals can be input terminals, output terminals, or bi-directional terminals. Each type of terminal has the following attributes:

1. Terminal name
2. Terminal width (LSB and MSB)
3. Terminal type

A terminal type could be scalar, vectored, or parameterized. Certain cases are illustrated here, to show what values the various attributes could take:

- Terminal: A

Name: A lsb=-1 msb=-1type=scalar

- Terminal: A<5..0>

Name: A lsb=-1 msb=-1type=vectored

- Terminal: A<size-1..0>

Name: A<size-1..0> lsb=-1 msb=-1type=parameterized

- **Net Bit Context**

Net bits have a bit and can return the complete net they represent. It is also possible to enumerate the instance terminals connected by this bit, and the aliases for this bit.
If the corresponding net is a scalar object, the bit represented by the net bit is -1.

- **Instance Terminal Bit Context**

Instance terminals have a bit and can return the complete instance terminal they represent. It is also possible to get the net bit connected to this instance terminal. If the corresponding instance terminal is a scalar object, the bit represented by the instance terminal bit is -1.

- **Connection Object**

An instance in a design is represented using an IInstance object.

- **Instance Terminal Context**

This is one instance terminal on an instance, for every terminal object on the corresponding cell interface.

Writing a TCL script with logical and physical view commands:

The following sequence can be followed for writing a Tcl script that uses logical or physical view commands:

1. Get Server Use commands such as sdaConn::getServer
2. Use server and find functions to get context to the design.
3. IDesign commands

Using the design context, you can access design properties, as well as instances and nets in the design.

General Syntax for Physical and Logical View Commands

This section describes the general syntax that can be followed for both logical and physical view commands. The commands can be used in one of the following two ways:

Method 1: Command Name with DataType

command <param> <param> <param> …For example:set dContextHandle [**lServer_loadDesign** $server $lib $cell "tbl_1"]

Method 2: Command with Datatype as Parameter

<param> command_modified <param> <param>set dContextHandle [$server **loadDesign** $lib $cell "tbl_1"]
The first parameter is $server, indicating that the loadDesign method is being used in the context of the server object specified by $server.
Notice that the value of the library and the cell to be loaded is passed as a variable using $ expression, whereas the value of the view to be loaded is specified in double quotes as tbl_1.
The difference between method 1 and method 2 is that in the second one, the command is modified to remove the data type from the command name. This helps increase the readability of the code.In the examples given, the command name has been modified from lServer_loadDesign
to loadDesign and the server information is passed as the first parameter before calling loadDesign.
For example: **lServer_loadDesign**
Method 1: set dContextHandle
[**lServer_loadDesign** $server $lib $cell "tbl_1"
]
Method 2: set dContextHandle
[$server **loadDesign** $lib $cell "tbl_1"
]
Here lServer_ is removed from the command. Therefore, the method name used is loadDesign and not lServer_loadDesign, as used in the previous example. Also, in this case, you need not pass $server as a parameter.

CConn_design

Returns the design handle for the specified connection. Use IInstTerm_beginConn to obtain the connection object.

Return Type

Design Handle

Syntax

```
CConn_design <CConn_Object>
```

Parameters

Parameter	Type	Description
`CConn_Object`	`Connection Object`	It specifies the object representing the connection interface in the design. This parameter is required.

Examples

```
CConn_design $CConn_Object
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

IInstTerm_beginConn

CConn_instTerm

Returns the instance terminal object, which is a part of the specified connection.

Return Type

Instance Terminal Object

Syntax

```
CConn_instTerm <CConn_Object>
```

Parameters

Parameter	Type	Description
`CConn_Object`	`Connection Object`	Specifies the object representing the connection interface in the design. This parameter is required.

Examples

```
CConn_instTerm $CConn_Object
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

IInstTerm_beginConn

CConn_net

Returns the net used for making the connection.

Return Type

Net Object

Syntax

```
CConn_net <CConn_Object>
```

Parameters

Parameter	Type	Description
`CConn_Object`	`Connection Object`	Specifies the object representing the connection interface in the design. This parameter is required.

Examples

```
CConn_net $CConn_Object
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

IInstTerm_beginConn

CConn_netLsb

Returns the LSB of the net making the connection. Note that this value could be less than the actual width of the net.

Return Type

LSB

Syntax

```
CConn_netLsb <CConn_Object>
```

Parameters

Parameter	Type	Description
`CConn_Object`	`Connectio Object`	Specifies the object representing the connection interface in the design. This parameter is required.

Examples

```
CConn_netLsb $CConn_Object
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

lInstTerm_beginConn

CConn_netMsb

Returns the MSB of the net making this connection. Note that this value could be less than the actual width of the net.

Return Type

MSB

Syntax

```
CConn_netMsb <CConn_Object>
```

Parameters

Parameter	Type	Description
`CConn_Object`	`Connection Object`	Specifies the object representing the connection interface in the design. This parameter is required.

Examples

```
CConn_netMsb $CConn_Object
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

lInstTerm_beginConn

CDesignContext_beginBase

This method gets the base iterator representing the start of base net bit enumeration for this design and all the levels below it.
It returns base net bit start iterator

Return Type

Base net bits iterator

Syntax

```
CDesignContext_beginBase <dContext_Handle>
```

Parameters

Parameter	Type	Description
`dContext_Handle`	`Design Context`	Context of the design This parameter is required.

Examples

```
CDesignContext_beginBase $dContext_Handle
```

Related Commands

getServer

IServer_findDesign

CDesignContext_cpath

This method returns the user canonical path for this object. The string returned is temporary, and would get destroyed with this object.
It returns the User canonical path

Return Type

Canonical Path

Syntax

```
CDesignContext_cpath <dContext_Handle>
```

Parameters

Parameter	Type	Description
`dContext_Handle`	`Design Context`	Context of the design This parameter is required.

Examples

```
CDesignContext_cpath $dContext_Handle
```

Related Commands

getServer

IServer_findDesign

CDesignContext_instance

Returns Instance context object for a given instance based on:
- Instance Object: Returns an instance context object based on the instance object provided.
- Instance Identifier: Returns an instance context object based on the instance identifier provided.

Return Type

Instance Context

Syntax

```
CDesignContext_instance <dContext_Handle> <p> | <id>
```

Parameters

Parameter	Type	Description
`dContext_Handle`	`Design Context`	Context of the design This parameter is required.
`p`	`IInstance object`	Instance pointer or object, whose context is required. This parameter is required.
`id`	`INT`	Identifier of instance whose context is needed. This parameter is required.

Examples

```
#CDesignContext_instance examples
#Example instance object
CDesignContext_instance $dContext_Handle $p
#Example instance identifier
CDesignContext_instance $dContext_Handle $id
```

Related Commands

getServer

IServer_findDesign

CDesignContext_net

Returns the net context for a given net based on:
- The Net Instance: Returns a net context object for the specified net object.
- The Net Identifier: Returns a net context object for the net with the specified identifier.

This command returns the net at the same level of the design and does not descend into the hierarchy.

Return Type

Net Context

Syntax

```
CDesignContext_net <dContext_Handle> <p> | <id>
```

Parameters

Parameter	Type	Description
`dContext_Handle`	`Design Context`	The context of the design. This parameter is required.
`p`	`Net Object`	The net object whose context is required. This parameter is required.
`id`	`INT`	The net identifier whose context is required. This parameter is required.

Examples

```
#Example of CDesignContext_net
#Example of net object
CDesignContext_net $dContext_Handle $p
#Example of identifier
CDesignContext_net $dContext_Handle $id
```

CDesignContext_netbit

This method returns a net bit context object, for the net bit provided.

Return Type

Net Bit Context

Syntax

```
CDesignContext_netbit <dContext_Handle> <id> <bitno>
```

Parameters

Parameter	Type	Description
`dContext_Handle`	`Design Context`	Context of the design This parameter is required.
`id`	`INT`	Identifier of net whose context is desired This parameter is required.
`bitno`	`INT`	bit of the net This parameter is required.

Examples

```
CDesignContext_netbit $dCxtHandle $id $bitno
```

Related Commands

getServer

IServer_findDesign

CDesignContext_spath

Use this method to get the system canonical path for the design object.
It returns the system canonical path as a string. The string is temporary, and is destroyed with the design object. Therefore, it is recommended that you save the canonical path in a separate variable

Return Type

Canonical Path

Syntax

```
CDesignContext_spath <dContext_Handle>
```

Parameters

Parameter	Type	Description
`dContext_Handle`	`Design Context`	Context of the design This parameter is required.

Examples

```
CDesignContext_spath $dCxtHandle
```

Related Commands

getServer

IServer_findDesign

CInstanceContext_containedDesign

Get Design for this instance. This method returns the design context object for this instance context. This is the only way to iterate on objects inside this instance in context. If this instance is not hierarchical, the design context object does not contain anything meaningful. Therefore, a check for valid design context objects should always be done following this method. This is done using the bool operator.

Return Type

Design Context

Syntax

```
CInstanceContext_containedDesign <CInstance_Context>
```

Parameters

Parameter	Type	Description
`CInstance_Context`	`Instance Context`	It specifies the context of the instance in the design This parameter is required.

Examples

```
CInstanceContext_containedDesign $CInstance_Context
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

CInstanceContext_design

Get parent design context. This method returns the design context, containing this instance context object.

Return Type

Parent Design Context

Syntax

```
CInstanceContext_design <CInstance_context>
```

Parameters

Parameter	Type	Description
`CInstance_context`	`Instance Context`	It specifies the context of the instance in the design This parameter is required.

Examples

```
CInstanceContext_design $CInstance_context
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

CInstanceContext_instTerm

Get Instance Terminal Context. This method returns an instance terminal context, created from the instance terminal object provided.

Return Type

Instance Terminal Context

Syntax

```
CInstanceContext_instTerm <CInstance_Context> <p>
```

Parameters

Parameter	Type	Description
`CInstance_Context`	`Instance Context`	It specifies the context of the instance in the design This parameter is required.
`p`	`INT`	Instance terminal whose context is required This parameter is required.

Examples

```
CInstanceContext_instTerm $CInstance_Context $p
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

CInstanceContext_physicalPackageName

Get the physical package name. This method returns the physical package name, of which this is the instance context. In case of instances of hierarchical blocks, empty is returned.

Return Type

Physical Package Name

Syntax

```
CInstanceContext_physicalPackageName <CInstance_Context>
```

Parameters

Parameter	Type	Description
`CInstance_Context`	`Instance Context`	It specifies the context of the instance in the design This parameter is required.

Examples

```
CInstanceContext_physicalPackageName $CInstance_Context
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

CInstanceContext_spath

Get the system canonical path. This method returns the system canonical path for this object. The string returned is temporary, and would get destroyed with this object.

Return Type

Canonical Path

Syntax

```
CInstanceContext_spath <CInstance_Context>
```

Parameters

Parameter	Type	Description
`CInstance_Context`	`Instance Context`	It specifies the context of the instance in the design This parameter is required.

Examples

```
CInstanceContext_spath $CInstance_Context
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

CInstTermBitContext_cpath

Get the user canonical path. This method returns the user canonical path for this object. The string returned is temporary, and would get destroyed with this object.

Return Type

Canonical Path

Syntax

```
CInstTermBitContext_cpath <CInstTermBitContext>
```

Parameters

Parameter	Type	Description
`CInstTermBitContext`	`Instance Terminal Bit Context`	It specifies the instance terminal bit context in the design This parameter is required.

Examples

```
CInstTermBitContext_cpath $CInstTermBitContext
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

CInstTermBitContext_instance

Get parent instance context. This method returns the instance context representing the parent of this instance terminal bit context.

Return Type

Instance Context

Syntax

```
CInstTermBitContext_instance <CInstTermBitContext>
```

Parameters

Parameter	Type	Description
`CInstTermBitContext`	`Instance Terminal Bit Context`	It specifies the instance terminal bit context in the design This parameter is required.

Examples

```
CInstTermBitContext_instance $CInstTermBitContext
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

CInstTermBitContext_netbitOuter

Get outer connected net bit. This method returns a net bit context connected to this instance terminal bit. The name outer is present because if the instance whose instance terminal this object is, happens to be a hierarchical instance, this method would return the net bit connected to the port outside that instance.

Return Type

Net bit context

Syntax

```
CInstTermBitContext_netbitOuter <CInstTermBitContext>
```

Parameters

Parameter	Type	Description
`CInstTermBitContext`	`Instance Terminal Bit COntext`	It specifies the instance terminal bit context in the design This parameter is required.

Examples

```
CInstTermBitContext_netbitOuter $CInstTermBitContext
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

CInstTermBitContext_physicalPinName

Get the physical pin name assigned to this pin. This method returns the physical pin number name assigned to this pin by the online packager. Returns physical pin number if assigned, 0 otherwise.

Return Type

Physical Pin Name

Syntax

```
CInstTermBitContext_physicalPinName <CInstTermBitContext>
```

Parameters

Parameter	Type	Description
`CInstTermBitContext`	`Instance Terminal Bit Context`	It specifies the instance terminal bit context in the design This parameter is required.

Examples

```
CInstTermBitContext_physicalPinName $CInstTermBitContext
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

CInstTermBitContext_spath

Get the system canonical path. This method returns the system canonical path for this object. The string returned is temporary, and would get destroyed with this object.

Return Type

Canonical Path

Syntax

```
CInstTermBitContext_spath <CInstTermBitContext>
```

Parameters

Parameter	Type	Description
`CInstTermBitContext`	`Instance Terminal Bit Context`	It specifies the instance terminal bit context in the design This parameter is required.

Examples

```
CInstTermBitContext_spath $CInstTermBitContext
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

CInstTermContext_beginBit

Returns the beginning iterator of the instance terminal bit context representing the instance terminal in the design.

Return Type

Terminal bit iterator

Syntax

```
CInstTermContext_beginBit <InstTermContext>
```

Parameters

Parameter	Type	Description
`InstTermContext`	`Instance Terminal Context`	Specifies the object representing the instance terminal in the design. This parameter is required.

Examples

```
CInstTermContext_beginBit $InstTermContext
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

CInstTermContext_cpath

Returns the user canonical path for the object.

Note: The string returned is temporary and is destroyed with this object. Therefore, the canonical path string must be stored in a separate variable.

Return Type

Canonical Path

Syntax

```
CInstTermContext_cpath <InstTermContext>
```

Parameters

Parameter	Type	Description
`InstTermContext`	`Instance Terminal Context`	Specifies the object representing the instance terminal in the design. This parameter is required.

Examples

```
CInstTermContext_cpath $InstTermContext
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

CInstTermContext_instance

Returns the instance context containing this instterm (instance terminal) context object.

Return Type

Instance Context

Syntax

```
CInstTermContext_instance <InstTermContext>
```

Parameters

Parameter	Type	Description
`InstTermContext`	`Instance Terminal Context`	Specifies the object representing the instance terminal in the design. This parameter is required.

Examples

```
CInstTermContext_instance $InstTermContext
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

CInstTermContext_spath

Returns the system canonical path for this object. The returned string is temporary and destroyed with this object.

Return Type

Canonical Path

Syntax

```
CInstTermContext_spath <InstTermContext>
```

Parameters

Parameter	Type	Description
`InstTermContext`	`Instance Terminal Context`	Specifies the object representing the instance terminal in the design. This parameter is required.

Examples

```
CInstTermContext_spath $InstTermContext
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

CNetBitContext_beginAlias

Begins alias iteration. This method returns an iterator representing the beginning of the aliased net bits for the specified net bit.

Return Type

Aliased net bit iterator

Syntax

```
CNetBitContext_beginAlias <Net_Bit_Context>
```

Parameters

Parameter	Type	Description
`Net_Bit_Context`	`Net Bit Context`	Specifies the context of the net bit in the design. This parameter is required.

Examples

```
CNetBitContext_beginAlias $Net_Bit_Context
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

IInstTerm_beginConn

CNetBitContext_cpath

Returns the user canonical path for this object.
Note: The string returned is temporary that gets destroyed with this object

Return Type

Canonical Path

Syntax

```
CNetBitContext_cpath <Net_Bit_Context>
```

Parameters

Parameter	Type	Description
`Net_Bit_Context`	`Net Bit Context`	Specifies the context of the net bit in the design. This parameter is required.

Examples

```
CNetBitContext_cpath $Net_Bit_Context
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

IInstTerm_beginConn

CNetBitContext_design

This method is used to get the parent design context.

Return Type

Design Context

Syntax

```
CNetBitContext_design <Net_Bit_Context>
```

Parameters

Parameter	Type	Description
`Net_Bit_Context`	`Net Bit Context`	Specifies the context of the net bit in the design. This parameter is required.

Examples

```
CNetBitContext_design $Net_Bit_Context
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

IInstTerm_beginConn

CNetBitContext_physicalNetName

Returns the physical net name assigned by the online packager to this net.

Return Type

Physical Net Name

Syntax

```
CNetBitContext_physicalNetName <Net_Bit_Context>
```

Parameters

Parameter	Type	Description
`Net_Bit_Context`	`Net Bit Context`	Specifies the context of the net bit in the design. This parameter is required.

Examples

```
CNetBitContext_physicalNetName $Net_Bit_Context
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

IInstTerm_beginConn

CNetBitContext_spath

Returns the system canonical path for the specified object. The string returned is temporary and gets destroyed with the object.

Return Type

Canonical Path

Syntax

```
CNetBitContext_spath <Net_Bit_Context>
```

Parameters

Parameter	Type	Description
`Net_Bit_Context`	`Net Bit Context`	Specifies the context of the net bit in the design. This parameter is required.

Examples

```
CNetBitContext_spath $Net_Bit_Context
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

IInstTerm_beginConn

CNetContext_bitContext

Returns the net bit context.

Note: This method works for vectored nets and is to be used with the isVectored function.

Return Type

Net bit context

Syntax

```
CNetContext_bitContext <CNet_Context> <bit>
```

Parameters

Parameter	Type	Description
`CNet_Context`	`Net Context`	Specifies the object representing the net in design. This parameter is required.
`bit`	`INT`	The bit number for which context is desired. This parameter is required.

Examples

```
CNetContext_bitContext $CNet_Context $bit
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

lDesign_cellInterface

lInstance_beginInstTerm

lInstTerm_beginConn

CNetContext_cpath

Returns the user canonical path for this object. The string returned is temporary and is destroyed with this object.

Return Type

Canonical Path

Syntax

```
CNetContext_cpath <CNet_Context>
```

Parameters

Parameter	Type	Description
`CNet_Context`	`Net Context`	Specifies the object representing the net in design This parameter is required.

Examples

```
CNetContext_cpath $CNet_Context
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

IInstTerm_beginConn

CNetContext_design

Returns the design context representing the parent of the specified net context.

Return Type

Design Context

Syntax

```
CNetContext_design <CNet_Context>
```

Parameters

Parameter	Type	Description
`CNet_Context`	`Net Context`	Specifies the object representing the net in the design. This parameter is required.

Examples

```
CNetContext_design $CNet_Context
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

IInstTerm_beginConn

CNetContext_spath

Returns the system canonical path for the specified object. The string returned is temporary and gets destroyed with the object.

Return Type

Canonical Path

Syntax

```
CNetContext_spath <CNet_Context>
```

Parameters

Parameter	Type	Description
`CNet_Context`	`Net Context`	Specifies the object representing the net in the design. This parameter is required.

Examples

```
#Example
CNetContext_spath $CNet_Context
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

lInstTerm_beginConn

getServer

Returns the handle to the connectivity server for the given TCL module.

Return Type

Server Handle

Syntax

```
sdaConn::getServer
```

Examples

```
set server [sdaConn::getServer]
```

ICellInterface_beginInstance

Returns an iterator representing the beginning of the instances of the specified cell interface.

Return Type

Instance iterator

Syntax

```
ICellInterface_beginInstance <ICell_Interface>
```

Parameters

Parameter	Type	Description
`ICell_Interface`	`Cell Interface Object`	Specifies the object representing the cell interface in the design. This parameter is required.

Examples

```
#Example
ICellInterface_beginInstance $ICell_Interface
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

lInstTerm_beginConn

ICellInterface_beginTerm

Returns an iterator representing the beginning of terminal enumerations for the specified cell interface. If there are no terminals for this cell interface, the begin iterator is equal to the end iterator.

Return Type

Terminal iterator

Syntax

```
ICellInterface_beginTerm <ICell_Interface>
```

Parameters

Parameter	Type	Description
`ICell_Interface`	`Cell Interface Object`	Specifies the object representing the cell interface in the design. This parameter is required.

Examples

```
#Example
ICellInterface_beginTerm $ICell_Interface
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

ICellInterface_findTerm

Finds the specified terminal in the cell interface based on the following parameters:

name: The return value is the iterator to the terminal object because multiple terminals of the same name are permissible in a cell interface. To ensure that all the terminals with the matching name are returned, an iterator to the found terminals is returned instead of individual terminal.

identifier: If the terminal is found, a pointer to the terminal object is returned, else 0 is returned.

Return Type

Terminal Object

Syntax

```
ICellInterface_findTerm <ICell_Interface> <name> | <id>
```

Parameters

Parameter	Type	Description
`ICell_Interface`	`Cell Interface Object`	Specifies the object representing the cell interface in the design. This parameter is required.
`name`	`String`	A string representing the name of the terminal to be found. This parameter is required.
`id`	`INT`	A nonzero positive integer representing the identifier of terminal to be found. This parameter is required.

Examples

```
#Examples of ICellInterface_findTerm
#Example - Name parameter
ICellInterface_findTerm $ICell_Interface $name
#Example - Identifier parameter
ICellInterface_findTerm $ICell_Interface $id
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

IInstTerm_beginConn

ICellInterface_lib

Returns the library the specified cell was loaded from.

Return Type

Library Name

Syntax

```
ICellInterface_lib <ICell_Interface>
```

Parameters

Parameter	Type	Description
`ICell_Interface`	`Cell Interface Object`	Specifies the object representing the cell interface in the design. This parameter is required.

Examples

```
#Example
ICellInterface_lib $ICell_Interface
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

ICellInterface_view

Returns the view the specified cell interface is bound to.

Return Type

View Name

Syntax

```
ICellInterface_view <ICell_Interface>
```

Parameters

Parameter	Type	Description
`ICell_Interface`	`Cell Interface Object`	Specifies the object representing the cell interface in the design. This parameter is required.

Examples

```
#Example
ICellInterface_view $ICell_Interface
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IDesign_beginAlias

This method returns an iterator representing the start of aliases in this design.

The IDesign class represents the hierarchical block. The interface of a design object is represented by its cell interface, which contains the interface port specification, library, cell, and view information.

A design can enumerate the following:
• Instances in the design
• Nets in the design
• Aliases in the design
• Connections in the design
• Cell interfaces used in the design

Return Type

Alias iterator

Syntax

```
IDesign_beginAlias <IDesign_object>
```

Parameters

Parameter	Type	Description
`IDesign_object`	`IDesign Object`	It specifies the object representing the design loaded on the connectivity server This parameter is required.

Related Commands

getServer

IServer_findDesign

IDesign_beginBase

This method returns an iterator representing the start of base net bits in this design.

Return Type

Base net bits iterator

Syntax

```
IDesign_beginBase <IDesign_object>
```

Parameters

Parameter	Type	Description
`IDesign_object`	`OBJECT`	It specifies the object representing the design loaded on the connectivity server This parameter is required.

Related Commands

getServer

IServer_findDesign

IDesign_beginConn

This method is used to get an iterator representing the start of the connections present in this design. This iterator skips the connections that are created as a part of a design template instantiation.

Return Type

Connection iterator

Syntax

```
IDesign_beginConn <IDesign_object>
```

Parameters

Parameter	Type	Description
`IDesign_object`	`IDesign Object`	It specifies the object representing the design loaded on the connectivity server This parameter is required.

Related Commands

getServer

IServer_findDesign

IDesign_beginGlobalNet

This method is used to get an iterator representing the start of the global nets available in this design.
It returns the iterator representing the start of the global net enumeration. The nets listed using the iterator, are a subset of all the nets available in the design.

Return Type

Net iterator

Syntax

```
IDesign_beginGlobalNet <IDesign_object>
```

Parameters

Parameter	Type	Description
`IDesign_object`	`IDesign Object`	It specifies the object representing the design loaded on the connectivity server This parameter is required.

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

This method is used to get an iterator representing the start of the instances used in this design. It returns an iterator representing the start of the instance listing.

Return Type

Instance iterator

Syntax

```
IDesign_beginInstance <IDesign_object>
```

Parameters

Parameter	Type	Description
`IDesign_object`	`IDesign Object`	It specifies the object representing the design loaded on the connectivity server This parameter is required.

Examples

```
IDesign_beginInstance $IDesign_object
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstanceIncludeTemplates

This method is used to get an iterator representing the start of instances used in this design. Use this method if you want the template instances to be included in the enumerations, such as bypass rails.
It returns an iterator representing the start of instance enumeration.

Return Type

Instance iterator

Syntax

```
IDesign_beginInstanceIncludeTemplates <IDesign_object>
```

Parameters

Parameter	Type	Description
`IDesign_object`	`IDesign Object`	It specifies the object representing the design loaded on the connectivity server This parameter is required.

Related Commands

getServer

IServer_findDesign

IDesign_beginNet

This method is used to get an iterator representing the start of nets used in this design. It returns an iterator representing start of net enumeration.

Return Type

Net iterator

Syntax

```
IDesign_beginNet <IDesign_object>
```

Parameters

Parameter	Type	Description
`IDesign_object`	`IDesign Object`	It specifies the object representing the design loaded on the connectivity server This parameter is required.

Examples

```
IDesign_beginNet $IDesign_object
```

Related Commands

getServer

IServer_findDesign

IDesign_beginNetIncludeTemplates

This method is used to obtain an iterator representing the start of the nets used in this design, including the nets created due to template instances.
It returns an iterator representing start of net enumeration.

Return Type

Net iterator

Syntax

```
IDesign_beginNetIncludeTemplates <IDesign_object>
```

Parameters

Parameter	Type	Description
`IDesign_object`	`IDesign Object`	It specifies the object representing the design loaded on the connectivity server This parameter is required.

Related Commands

getServer

IServer_findDesign

IDesign_cellInterface

The cellInterface method is used to get a cell interface object representing the interface characteristics of this design.
It returns the Cell interface for this design.

Return Type

Cell Interface Object

Syntax

```
IDesign_cellInterface <IDesign_object>
```

Parameters

Parameter	Type	Description
`IDesign_object`	`IDesign Object`	It specifies the object representing the design loaded on the connectivity server This parameter is required.

Examples

```
IDesign_cellInterface $IDesign_object
```

Related Commands

getServer

IServer_findDesign

IDesign_findCellInterface

Finds a cell interface in the design. Depending on the parameters passed, this command can be used to:

- Find interface specified by an identifier: The cell interface identifier is passed as a parameter to the findCellInterface method.

- Find cell interface of specified lib:cell:view: The required lib:cell:view is passed as a parameter to the findCellInterface method.

Return Type

Cell Interface Object

Syntax

```
IDesign_findCellInterface <IDesign_Object> <id> | <lib> <cell> <view>
```

Parameters

Parameter	Type	Description
`IDesign_object`	`IDesign Object`	Represents the design loaded on the connectivity server This parameter is required.
`id`	`INT`	Cell interface identifier represented by a nonzero positive integer. This parameter is required.
`lib`	`String`	Specifies the Library name of the cell to be found. This parameter is required.
`cell`	`String`	Specifies the name of the cell to be found. This parameter is required.
`view`	`String`	Specifies the view of the cell to be found. This parameter is required.

Examples

```
#IDesign_findCellInterface examples

#Find interface specified by an identifier
IDesign_findCellInterface $IDesign_Object $id

#Find cell interface of specified lib:cell:view
IDesign_findCellInterface $IDesign_Object $lib $cell $view
```

Related Commands

getServer

IServer_findDesign

IDesign_findNet

Searches for a net based on a name or an identifier. If found, returns the net object. Else, returns NULL.

Return Type

Net Object

Syntax

```
IDesign_findNet <IDesign_object> <name> | <id>
```

Parameters

Parameter	Type	Description
`IDesign_object`	`IDesign Object`	Specifies the object representing the design loaded on the connectivity server. This parameter is required.
`name`	`String`	Name of the net This parameter is required.
`id`	`INT`	Identifier of the net This parameter is required.

Examples

```
#Examples of IDesign_findNet command

#Example of search based on a name.
IDesign_findNet $IDesign_Object $name

#Example of search based on an identifier.
Example 2: IDesign_findNet $IDesign_Object $id
```

Related Commands

getServer

IServer_findDesign

IDesign_isModified

This command checks if modifications exist in the design since last save.
This method returns true if the design was modified since it was last saved.

Return Type

INT

Syntax

```
IDesign_isModified <IDesign_object>
```

Parameters

Parameter	Type	Description
`IDesign_object`	`IDesign Object`	It specifies the object representing the design loaded on the connectivity server This parameter is required.

Examples

```
IDesign_isModified $IDesign_object
```

Related Commands

getServer

IServer_findDesign

IDesign_server

This method gets the server object, if any, which loaded this design.
This method returns the Schema object from which this design was loaded.

Return Type

Server Object

Syntax

```
IDesign_server <IDesign_object>
```

Parameters

Parameter	Type	Description
`IDesign_object`	`IDesign Object`	It specifies the object representing the design loaded on the connectivity server This parameter is required.

Examples

```
IDesign_server $IDesign_object
```

Related Commands

getServer

IServer_findDesign

IInstance_beginInstTerm

This method is used to get an iterator representing the start of instance terminal enumerations for this instance. If there are no instance terminals, this iterator is equal to the end iterator.

Return Type

Instance Terminal Iterator

Syntax

```
IInstance_beginInstTerm <IInstance_Object>
```

Parameters

Parameter	Type	Description
`IInstance_Object`	`IInstance Object`	It specifies the object representing an instance in the design This parameter is required.

Examples

```
IInstance_beginInstTerm $IInstance_Object
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_cellInterface

Returns the cell interface of which the specified instance is an object.

An instance in a design is represented by an IInstance object. Instances are instantiations of a particular cell interface object. In addition, instances can contain values for parameters specified with the cell interface. Instances also enumerate the instance terminals on themselves.

Return Type

Cell Interface Object

Syntax

```
IInstance_cellInterface <IInstance_Object>
```

Parameters

Parameter	Type	Description
`IInstance_Object`	`IInstance Object`	Specifies the object representing an instance in the design. This parameter is required.

Examples

```
#Example
IInstance_cellInterface $IInstance_Object
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

lDesign_cellInterface

IInstance_containedDesign

This method is used to get the design represented by this instance. It returns design pointer representing this instance. If this instance is a primitive and not a hierarchical block, NULL is returned.

Return Type

Design Context

Syntax

```
IInstance_containedDesign <IInstance_Object>
```

Parameters

Parameter	Type	Description
`IInstance_Object`	`IInstance Object`	It specifies the object representing an instance in the design This parameter is required.

Examples

```
IInstance_containedDesign $IInstance_Object
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstTerm_beginBit

Returns the instance terminal bit begin iterator. The iterator represents the start of instance terminal bits for the specified instance terminal.

Return Type

Terminal bit iterator

Syntax

```
IInstTerm_beginBit <IInst_Term>
```

Parameters

Parameter	Type	Description
`IInst_Term`	`Instance Terminal Context`	Specifies the object representing the instance terminal in the design This parameter is required.

Examples

```
#Example
IInstTerm_beginBit $IInst_Term
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

lInstTerm_beginConn

Returns the connection begin iterator. This iterator represents the start of connections made by the specified instance terminal.

Return Type

Connection iterator

Syntax

```
lInstTerm_beginConn <lInst_Term>
```

Parameters

Parameter	Type	Description
`lInst_Term`	`Instance terminal context`	Specifies the object representing the instance terminal in the design. This parameter is required.

Examples

```
#Example
lInstTerm_beginConn $lInst_Term
```

Related Commands

getServer

lServer_findDesign

lDesign_beginInstance

lDesign_cellInterface

lInstance_beginInstTerm

IInstTerm_instance

Returns the instance that has the current instantiation of the terminal.

Return Type

Instance Context

Syntax

```
IInstTerm_instance <IInst_Term>
```

Parameters

Parameter	Type	Description
`IInst_Term`	`Instance Terminal Context`	Specifies the object representing the instance terminal in the design This parameter is required.

Examples

```
#Example
IInstTerm_instance $IInst_Term
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

lInstTerm_term

Returns the terminal object of which the current instterm (instance terminal) object is an instance.

Return Type

Terminal Object

Syntax

```
IInstTerm_term <IInst_Term>
```

Parameters

Parameter	Type	Description
`IInst_Term`	`Instance Terminal Context`	Specifies the object representing the instance terminal in the design This parameter is required.

Examples

```
#Example
IInstTerm_term $IInst_Term
```

Related Commands

getServer

lServer_findDesign

lDesign_beginInstance

lDesign_cellInterface

lInstance_beginInstTerm

INet_beginBit

This method is used to get the net bit begin iterator, which represents the start of net bits for this net. In case of a scalar net, a net bit object is returned, with its bit information set to -1.

Return Type

Net bit iterator

Syntax

```
INet_beginBit <INet>
```

Parameters

Parameter	Type	Description
`INet`	`Net Context`	It specifies the object representing the net in design This parameter is required.

Examples

```
INet_beginBit $INet
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

IInstTerm_beginConn

INet_beginConn

This method is used to get the connection begin iterator. This iterator represents the start of the connections made by this net. It does not return the connections made to instances that form part of a design template.

Return Type

Connection iterator

Syntax

```
INet_beginConn <INet>
```

Parameters

Parameter	Type	Description
`INet`	`Net Context`	It specifies the object representing the net in design This parameter is required.

Examples

```
INet_beginConn $INet
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

lInstTerm_beginConn

INet_inheritedGlobal

This method checks if this global signal is inherited from one of the sub-blocks of this design. A global signal is said to be inherited if it directly, or indirectly, over-rides a global signal in one of its sub-blocks. This call is valid only for global signals.
Returns the global net inheritance attribute. A true value is returned if the global net is inherited, and a false value is returned if global net is not inherited.

Return Type

INT

Syntax

```
INet_inheritedGlobal <INet>
```

Parameters

Parameter	Type	Description
`INet`	`NetContext`	It specifies the object representing the net in design This parameter is required.

Examples

```
INet_inheritedGlobal $INet
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

lInstance_beginInstTerm

lInstTerm_beginConn

INet_isScalar

This method checks if the net is scalar. Returns true if the net is scalar, else returns false.

Return Type

INT

Syntax

```
INet_isScalar <INet>
```

Parameters

Parameter	Type	Description
`INet`	`Net Context`	It specifies the object representing the net in design This parameter is required.

Examples

```
INet_isScalar $INet
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

IInstTerm_beginConn

INet_isVectored

Checks if the net is vectored. A True value is returned for a vectored net.

Return Type

INT

Syntax

```
INet_isVectored <INet>
```

Parameters

Parameter	Type	Description
`INet`	`Net Context`	Specifies the object representing the net in design. This parameter is required.

Examples

```
#Example
INet_isVectored $INet
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

IInstTerm_beginConn

IServer_beginDesignContext

This method returns an iterator representing the start of design enumerations loaded in this server.
Note that there is no specific order of iteration.
It returns Design context begin iterator.

Return Type

Design context iterator

Syntax

```
IServer_beginDesignContext <server>
```

Parameters

Parameter	Type	Description
`server`	`OBJECT`	It specifies the handle to the connectivity server This parameter is required.

Examples

```
IServer_beginDesignContext $server
```

Related Commands

getServer

IServer_findDesign

Use this method to find the specified design from the list of designs loaded in the server. Returns the context of the design if the specified design name is found.

Return Type

Design Context

Syntax

```
IServer_findDesign <server> <design_name>
```

Parameters

Parameter	Type	Description
`server`	`OBJECT`	It specifies the handle to the connectivity server This parameter is required.
`design_name`	`STRING`	Name of the design This parameter is required.

Examples

```
IServer_findDesign $server $designName
```

Related Commands

getServer

IServer_findFromCpath

Use this method to find an object from the user canonical path.
It returns the canonical path container representing the object found.

Important:
The returned pointer is allocated by the connectivity server and should be freed by the user by making a delete call. To support auto destruction, it is recommended that you return context object, instead of pointer.

Return Type

Canonical Path container

Syntax

```
IServer_findFromCpath <server> <path>
```

Parameters

Parameter	Type	Description
`server`	`OBJECT`	It specifies the handle to the connectivity server This parameter is required.
`path`	`STRING`	It specifies the user path of the object to be located. This parameter is required.

Examples

```
IServer_findFromCpath $server $path
```

Related Commands

getServer

IServer_findFromSpath

Use this method to find an object from the system canonical path.
Returns the canonical path container representing the found object.

Important:
The returned pointer is allocated by the connectivity server and should be freed by the user by making a delete call. To support auto-destruction, remember to return the context object, instead of the pointer

Return Type

Canonical Path container

Syntax

```
IServer_findFromSpath <server> <path>
```

Parameters

Parameter	Type	Description
`server`	`OBJECT`	It specifies the handle to the connectivity server This parameter is required.
`path`	`STRING`	It specifies the system path of the object to be located This parameter is required.

Examples

```
IServer_findFromSpath $server $path
```

Related Commands

getServer

IServer_findOccurrences

This method is used for finding all the occurrences of the specified design name. Returns the iterator representing all the occurrences of the design.

Return Type

Design Occurrence Iterator

Syntax

```
IServer_findOccurrences <server> <design_name>
```

Parameters

Parameter	Type	Description
`server`	`OBJECT`	It specifies the handle to the connectivity server This parameter is required.
`design_name`	`STRING`	Name of the design This parameter is required.

Examples

```
IServer_findOccurrences $server $designName
```

Related Commands

getServer

IServer_loadDesign

This method loads a design block in server. If the design already exists in memory, that design is returned.
It returns an IDesign object representing the design loaded on the connectivity server.

Return Type

IDesign Object

Syntax

```
IServer_loadDesign <server> <library> <cell> <view>
```

Parameters

Parameter	Type	Description
`server`	`OBJECT`	It specifies the handle to the connectivity server This parameter is required.
`library`	`STRING`	It specifies the name of the library, in filesystem namespace, in which new design is created This parameter is required.
`cell`	`STRING`	It specifies the name of the design cell This parameter is required.
`view`	`STRING`	It specifies the view of the design in filesys namespace This parameter is required.

Examples

```
Example1:
set dContextHandle [lServer_loadDesign $server $lib $cell "tbl_1"]
In the preceding example, the lServer_loadDesign method is used for the loading the
tbl_1 view of the specified design.
The first parameter is $server, indicating that the loadDesign method is being used in
the context to the server object specified by $server. Notice that the value of the
library and the cell to be loaded is passed as a variable using $ expression, whereas
the value of the view to be loaded is specified in double quotes as tbl_1.
An alternate method of using the loadDesign method is shown in the next example.

Example2:
set dContextHandle [$server loadDesign $lib $cell "tbl_1"]
In this example, the loadDesign method is used in the context of the $server object.
Therefore, the method name used is loadDesign and not lServer_loadDesign, as used in
the previous example. Also, in this case you need not pass $server as a parameter.

Important
In both the preceding examples, $lib and $cell indicated that the values of these
arguments are passed as parameters. The value of the third parameter is tbl_1 and is,
therefore, specified in double quotes.
```

Related Commands

getServer

lTerm_beginBit

Returns an iterator representing the start of the bit terminals for the specified terminal.

Return Type

Terminal bit iterator

Syntax

```
lTerm_beginBit <lTerm>
```

Parameters

Parameter	Type	Description
`lTerm`	`Terminal Context`	Specifies the object representing the terminal in design. This parameter is required.

Examples

```
#Example
lTerm_beginBit $lTerm
```

Related Commands

getServer

lServer_findDesign

lDesign_beginInstance

lDesign_cellInterface

lInstance_beginInstTerm

ITerm_instance

Returns the parent instance object. This object represents the parent of the specified ITerm (instance terminal) object.

Return Type

Instance Context

Syntax

```
ITerm_instance <ITerm>
```

Parameters

Parameter	Type	Description
`ITerm`	`Terminal Context`	Specifies the object representing the terminal in the design. This parameter is required.

Examples

```
#Example
ITerm_instance $ITerm
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

ITerm_name

Returns the name of terminal. The name of a terminal is the base name, which is stripped of the vector or parameterized notations.
For example, the base names of A<5..0> and A<size-1..0> is A.

Return Type

Terminal Name

Syntax

```
ITerm_name <ITerm>
```

Parameters

Parameter	Type	Description
`ITerm`	`Terminal Context`	Specifies the object representing the terminal in design. This parameter is required.

Examples

```
#Example
ITerm_name $ITerm
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

ITerm_vectorName

This method is used to get vector name of terminal

Return Type

Vector Name

Syntax

```
ITerm_vectorName <ITerm>
```

Parameters

Parameter	Type	Description
`ITerm`	`Terminal Context`	It specifies the object representing the terminal in design This parameter is required.

Examples

```
ITerm_vectorName $ITerm
```

Related Commands

getServer

IServer_findDesign

IDesign_beginInstance

IDesign_cellInterface

IInstance_beginInstTerm

5

Object Manipulation

addHyperlink

Hyperlinks are supported on the following items:
1. Note that is RICH_NOTE
2. Table that is TABLE
3. Occurrence property of a component that is OCCURRENCE_PROP

If a type of the object on which a hyperlink is being added is RICH NOTE then the start and end position needs to be provided.
TABLE - Provide cell number, start, and end position of the text.
OCCURRENCE_PROP - Name of occurrence property.

Hyperlink address can be:
1. A web page URL, such as {http://www.google.com}
2. An e-mail address, such as {mailto:wsw?subject=swsw}
3. A schematic Page, such as {proj:@worklib.sddwd1(tbl_1)?pageuid=3}

Return Type

NONE

Syntax

```
addHyperlink -type <item_type> ?-cell <row_no>,<column_no>? ?-posStart <start_position>
-posEnd <end_position>? | -n <property_name> | -posStart <start_position> -posEnd
<end_position> -t <hyperlink_text> -a <hyperlink_address>
```

Parameters

Parameter	Type	Description
`item_type`	`STRING`	Supported item types are RICH_NOTE,TABLE, and OCCURRENCE_PROP. This parameter is required.
`row_no`	`INT`	Specify the row number of the table on which hyperlink needs to be added. Applicable if item_type is Table This parameter is optional.
`column_no`	`INT`	Specify the column number of the table on which hyperlink needs to be added. Applicable if item_type is Table This parameter is optional.
`start_position`	`INT`	Position from where the hyperlink text starts. Start position should start from 0 and It goes to (number of character -1) in the item. Start position should be less than or equal to end position. Applicable if item_type is Table or RICH_NOTE This parameter is optional.
`end_position`	`INT`	Position from where the hyperlink text ends. End position should start from 0 and It goes to (number of character -1) in the item. End position should be greater than or equal to start position. Applicable if item_type is Table or RICH_NOTE This parameter is optional.
`property_name`	`STRING`	Specify the name of occurrence property on which hyperlink needs to be added. Applicable if item_type is a OCCURRENCE_PROP This parameter is optional.
`hyperlink_text`	`STRING`	Specify the text on which hyperlink will be shown This parameter is required.
`hyperlink_address`	`STRING`	Specify a hyperlink address. Hyperlink address can be a web page, email address or schematic page This parameter is required.

Examples

```
#adds a hyperlink to a web url on note.
addHyperlink -type RICH_NOTE -posStart 1 -posEnd 7 -t {google} -a
{http://www.google.com}
#adds a hyperlink to email on a cell of table.
addHyperlink -type TABLE -cell {0,0} -posStart 0 -posEnd 5 -t {Hyper} -a
{mailto:xyz@gmail.com?subject=Adding Hyperlink}
#adds a hyperlink to a web url on an occurrence property of a component.
addHyperlink -type OCCURANCE_PROP -name {ABC} -t {XYZ} -a {http://www.google.com}
#adds a hyperlink to a schematic page on a cell of table.
addHyperlink -type TABLE -cell {0,0} -posStart 0 -posEnd 5 -t {Hyper} -a
{proj:@worklib.sddwd1(tbl_1)?pageuid=7}
```

Related Commands

removeHyperlink

editHyperlink

openHyperlink

align

Aligns the selected elements on the schematic page.

Return Type

NONE

Syntax

```
align <type>
```

Parameters

Parameter	Type	Description
`type`	`INT`	Type of component selected This parameter is required.

bringForward

Brings the selected element one level forward on the canvas.

Return Type

NONE

Syntax

```
bringForward
```

Examples

```
bringForward
```

bringToFront

Brings the element to the front of all the elements on the schematic page.

Return Type

INT

Syntax

```
bringToFront
```

Examples

```
bringToFront
```

changeByPassCapQuantity

Use changeByPassCapQuantity to change the quality of the bypass capacitors. It works on selected bypass rail. Select any bypass rail and trigger sch::changeByPassCapQuantity command. UI will come up to update quantity of bypass capacitors.

Return Type

NONE

Syntax

```
sch::changeByPassCapQuantity
```

Examples

```
#select any bypass rail and use the below command:
sch::changeByPassCapQuantity
#select new quantity from UI and press Ok.
#quantity of capacitors will get modified.
```

Related Commands

addBypass

changeByPassParentDistance

Use changeByPassParentDistance to change the bypass parent component distance property. It works on selected bypass rail. Select any bypass rail and trigger sch::changeByPassParentDistance command. UI will come up to update distance value along with units.

Return Type

NONE

Syntax

```
sch::changeByPassParentDistance
```

Examples

```
#select bypass rail and use below command:
sch::changeByPassParentDistance
#UI will come up. Update distance value/units and press ok.
```

clearNetAsBaseNet

This command will remove the Base Net attribute from the selected Nets. It needs to have a net selected on canvas to work.
This command comes in use mostly between aliased nets in which there is one winning net among all the aliased nets. To remove the base Net attribute from the Base net, this command is run after selecting the candidate net on the canvas. Then after running this command for the selected net, that base Net attribute of net will be clear.

Return Type

BOOL

Syntax

```
clearNetAsBaseNet
```

Examples

```
clearNetAsBaseNet
```

closeViewsforBlock

This command will close all the open views of the specified block. If there is no view open, there will be no result or response.

Return Type

NONE

Syntax

```
sch::closeViewsforBlock <libName> <blockName>
```

Parameters

Parameter	Type	Description
`libName`	`STRING`	Name of library from which the block is loaded. This parameter is required.
`blockName`	`STRING`	Name of block for which views needs to be closed. This parameter is required.

Examples

```
#command to close all views of 'block_1' from the 'lib_1' library
sch::closeViewsforBlock lib_1 block_1
```

copy

Copies an element to the clipboard.

Return Type

NONE

Syntax

```
copy type <type_of_object> <x-coordinate> <y-coordinate>
```

Parameters

Parameter	Type	Description
`type_of_object`	`STRING`	The object type. This value must be specified in uppercase. This parameter is required.
`x-coordinate`	`INT`	The X-coordinate of the object being copied. This parameter is required.
`y-coordinate`	`INT`	The Y-coordinate of the object being copied. This parameter is required.

Examples

```
copy type ELLIPSE 8081 4040
```

cut

Cuts the selected object from the canvas and places it on the clipboard. The object must be selected before running the command.

Return Type

BOOLEAN

Syntax

```
cut
```

Examples

```
cut
```

dbGetRotationValue

It returns the current rotation of an item, in degrees (0.0, 90.0, 270.0, 360.0).

Return Type

DOUBLE

Syntax

```
sch::dbGetRotationValue <dbId>
```

Parameters

Parameter	Type	Description
`dbId`	`INT`	dbID of the item to be queried This parameter is required.

Examples

```
#get the current active page
set currentActivePage [sch::dbGetActivePage]
#get the current selected items
set currentlySelectedItem [sch::dbGetSelectedItems $currentActivePage]
#loop over the dbIds for the currently selected items, get the rotation value,
#and print it out
foreach item $currentlySelectedItem { set rotationValue [sch::dbGetRotationValue
$item]; puts "Rotation for item $item is $rotationValue "}
```

Related Commands

dbGetRotationValueOfProp

deleteTableColumn

Use deleteTableColumn to delete the column(s) of a table starting from a specific column number.

Return Type

NONE

Syntax

```
deleteTableColumn <columnNumber> <numberofColumns>
```

Parameters

Parameter	Type	Description
`columnNumber`	`INTEGER`	The column number starting from which columns are to be deleted. Column Number should start from 1 and it can go up to the no of columns of the table. This parameter is required.
`numberofColumns`	`INTEGER`	The number of columns to be deleted. This parameter is required.

Examples

```
deleteTableColumn 4 3
```

deleteTableRow

Use deleteTableRow to delete the rows(s) of a table starting from a specific row number.

Return Type

INT

Syntax

```
deleteTableRow <rowNumber> <numberofRows>
```

Parameters

Parameter	Type	Description
`rowNumber`	`INT`	The row number starting from which the rows are to be deleted. Row Number should start from 1 and it can go up to the no of rows of the table. This parameter is required.
`numberofRows`	`INT`	The number of rows to be deleted. This parameter is required.

Examples

```
deleteTableRow 4 3
```

distribute

Evenly places the selected elements on the canvas, either vertically or horizontally. At least, three elements should be selected.

Return Type

NONE

Syntax

```
distribute <type>
```

Parameters

Parameter	Type	Description
`type`	`STRING`	Type of orientation. The valid values are horizontal and vertical. This parameter is required.

Examples

```
#Command to place the selected elements horizontally
distribute horizontal

#Command to place the selected elements vertically
distribute vertical
```

drawStubs

Draws wires on all the unconnected pins of the selected component(s).

Return Type

NONE

Syntax

```
drawStubs
```

Examples

```
drawStubs
```

editHyperlink

Use editHyperlink to edit an existing hyperlink.

Currently, the hyperlink is supported on the following items:
1. Note that is RICH_NOTE
2. Table that is TABLE
3. Occurrence property of a component that is OCCURRENCE_PROP

If a type is:
RICH_NOTE, then it provides the starting and the ending position.
TABLE, then it provides the cell number, the starting and the end position of the text.
OCCURRENCE_PROP, then the name of the occurrence property.

Hyperlink address can be:
1. A web page URL, such as {http://www.google.com}
2. An e-mail address, such as {mailto:wsw?subject=swsw}
3. A schematic Page, such as {proj:@worklib.sddwd1(tbl_1)?pageuid=3}

Return Type

NONE

Syntax

```
editHyperlink -type <item_type> -cell <row_no>,<column_no> -posStart <start_position> -
posEnd <end_position> | -n <property_name> | -posStart <start_position> -posEnd
<end_position> -t <hyperlink_text> -a <hyperlink_address>
```

Parameters

Parameter	Type	Description
`item_type`	`STRING`	Supported item types are RICH_NOTE,TABLE and OCCURRENCE_PROP. This parameter is required.
`row_no`	`INT`	Specify the row number of the table on which hyperlink needs to be added. Applicable if item_type is Table This parameter is optional.
`column_no`	`INT`	Specify the column number of the table on which hyperlink needs to be added. Applicable if item_type is Table This parameter is optional.
`start_position`	`INT`	Position from where the hyperlink text starts. Start position should start from 0 and It goes to (number of character -1) in the item. Start position should be less than or equal to end position. Applicable if item_type is Table or RICH_NOTE This parameter is optional.
`end_position`	`INT`	Position from where the hyperlink text ends. End position should start from 0 and It goes to (number of character -1) in the item. End position should be greater than or equal to start position. Applicable if item_type is Table or RICH_NOTE This parameter is optional.
`property_name`	`STRING`	Specify the name of occurrence property on which hyperlink needs to be added. Applicable if item_type is a OCCURRENCE_PROP This parameter is optional.
`hyperlink_text`	`STRING`	Specify the text on which hyperlink will be shown This parameter is required.
`hyperlink_address`	`STRING`	Specify a hyperlink address. Hyperlink address can be a web page, email address or schematic page This parameter is required.

Examples

```
#edit a hyperlink to a web url on note
editHyperlink -type RICH_NOTE -posStart 1 -posEnd 7 -t {google} -a
{http://www.google.com}
#edit a hyperlink to email on a cell of table
editHyperlink -type TABLE -cell {0,0} -posStart 0 -posEnd 5 -t {Hyper} -a
{mailto:xyz@gmail.com?subject=Adding Hyperlink}
#edit a hyperlink to a web url on an occurrence property of a component
editHyperlink -type OCCURANCE_PROP -name {ABC} -t {XYZ} -a {http://www.google.com}
#edit a hyperlink to a schematic page on a cell of table
editHyperlink -type TABLE -cell {0,0} -posStart 0 -posEnd 5 -t {Hyper} -a
{proj:@worklib.sddwd1(tbl_1)?pageuid=7}
```

editTable

Commands to set whether a table item is to be editable or not. Needs to have table item selected on canvas before running the command.

Return Type

NONE

Syntax

```
editTable <allowEdit>
```

Parameters

Parameter	Type	Description
`allowEdit`	`BOOL`	specify if table is to be editable by values true or false This parameter is required.

Examples

```
#To change editable property of complete table
editTable true
editTable false
```

editTableCells

Controls whether cells can be edited. If editing cells is disabled, the content in those cells cannot be changed. However, the formatting and styles, such as font name and font size can be changed.

Return Type

NULL

Syntax

```
editTableCells <isEditable> <startRow> <startColumn> <endRow> <endColumn>
```

Parameters

Parameter	Type	Description
`isEditable`	`BOOL`	Changes the editable property. If false then property cannot be changed and if true then property can be changed. This parameter is required.
`startRow`	`INT`	Row number from where changing the editable property for multiple cells starts This parameter is required.
`startColumn`	`INT`	Column number from where changing the editable property for multiple cells starts This parameter is required.
`endRow`	`INT`	Row number where changing the editable property for multiple cells ends This parameter is required.
`endColumn`	`INT`	Column number where changing the editable property for multiple cells ends This parameter is required.

Examples

```
#Command to disable content changes of multiple cells from 1,1 to 4,4.
editTableCells false 1 1 4 4
```

Related Commands

editTableRow

editTableColumn

editTableColumn

Controls whether a column of a table can be edited. If the editing of a column is disabled, the content in that column cannot be changed. However, the formatting and styles, such as font name and font size can be changed.

Return Type

NULL

Syntax

```
editTableColumn <isEditable> <column>
```

Parameters

Parameter	Type	Description
`isEditable`	`BOOL`	Changes the editable property of column. If false then property cannot be changed and if true then property can be changed. This parameter is required.
`column`	`INT`	The column number This parameter is required.

Examples

```
#Command to disable content changes in the third column
editTableColumn false 3
```

Related Commands

editTableRow

editTableCells

editTableHeader

Commands to set whether the header of a table item is to be editable or not. Needs to have table item selected on canvas before running the command.

Return Type

NONE

Syntax

```
editTableHeader <allowEdit>
```

Parameters

Parameter	Type	Description
`allowEdit`	`BOOL`	specify if table header is to be editable by values true or false This parameter is required.

Examples

```
#Command to make table header editable, select table item and give command
editTable true
```

editTableRow

Controls whether a row of a table can be edited. If the editing of a row is disabled, the content in that row cannot be changed. However, the formatting and styles, such as font name and font size can be changed.

Return Type

NULL

Syntax

```
editTableRow <isEditable> <row>
```

Parameters

Parameter	Type	Description
`isEditable`	`BOOL`	Changes the editable property of row. If false then property cannot be changed and if true then property can be changed. This parameter is required.
`row`	`INT`	Row number whose editing is being modified This parameter is required.

Examples

```
#Command to disable content changes to the first row
editTableRow false 1
```

Related Commands

editTableColumn

editTableCells

getBlockFolderPath

It returns the complete path to the layout, package, BOM, and external layout folder for a specified block in the project.

Return Type

STRING

Syntax

```
cps::getBlockFolderPath <directiveName> <blockName>
```

Parameters

Parameter	Type	Description
`directiveName`	`STRING`	Name of the predefined directives. "physical_folder" - defines the layout folder, "packaged_folder" - defines where package files (*.pst) would be created, "bom_folder" - defines where BOM reports would be created, and "external_allegro_board_folder" - defines where external layout files would be created This parameter is required.
`blockName`	`STRING`	Name of the block for which the path needs to be computed This parameter is required.

Examples

```
#print the "physical_folder" path for the block named "labrun"
puts "[cps::getBlockFolderPath physical_folder labrun]"
#output - /home/user1/workshop/projects/labrun/output/labrun/physical
```

group

Forms a group of selected elements on the schematic page. At least two elements are required to be selected. This command is preceded by the selectObject command.

Return Type

INT

Syntax

```
group
```

Examples

```
group
```

highlightObject

Highlights an object on the canvas.

Return Type

NONE

Syntax

```
highLightObject [<top_x-coordinate> <top_y-coordinate> <bottom_x-coordinate> <bottom_y-coordinate>] [mode] [-spath <path>] [<page_path>]
```

Parameters

Parameter	Type	Description
`[<top_x-coordinate> <top_y-coordinate> <bottom_x-coordinate> <bottom_y-coordinate>]`	`INT`	Grid coordinates of the component. This parameter is optional.
`[mode]`	`STRING`	Selection mode. This parameter is optional.
`[-sPath <path>]`	`STRING`	The spath (lib.cell:view) of the object. This parameter is required.
`[<page_path>]`	`STRING`	The path to the page. This parameter is required.

Examples

```
highlightObject -sPath @worklib.workshop1(tbl_1):page(1)
```

insertTableColumnLeft

Use insertTableColumnLeft to insert column(s) to the left of a specified column in the table.

Return Type

INT

Syntax

```
insertColumnLeft <columnNumber> <numberofcolumns>
```

Parameters

Parameter	Type	Description
`columnNumber`	`INT`	The column number to the left of which columns are to be inserted. Column Number should start from 1 and it can go up to the no of columns of the table. This parameter is required.
`numberofcolumns`	`INT`	The number of columns to be inserted. This parameter is required.

Examples

```
insertTableColumnLeft 4 3
#3 columns will be inserted before column number 4.
```

insertTableColumnRight

Use insertTableColumnRight to insert column(s) to the right of a specific column in the table.

Return Type

NONE

Syntax

```
insertTableColumnRight <columnNumber> <numberofcolumns>
```

Parameters

Parameter	Type	Description
`columnNumber`	`INT`	The column number to the right of which columns are to be inserted. Column Number should start from 1 and it can go up to the no of columns of the table. This parameter is required.
`numberofcolumns`	`INT`	The number of columns to be inserted. This parameter is required.

Examples

```
insertTableColumnRight 3 4
#4 columns will be inserted after column number 3
```

insertTableRowAbove

Use insertTableRowAbove to insert row(s) in a table above a specific row.

Return Type

NONE

Syntax

```
insertTableRowAbove <rowNumber> <numberofRows>
```

Parameters

Parameter	Type	Description
`rowNumber`	`INT`	The row number above which new row(s) are to be added.Row Number should start from 1 and it can go up to the no of rows of the table. This parameter is required.
`numberofRows`	`INT`	The number of rows to be inserted. This parameter is required.

Examples

```
insertTableRowAbove 1 2
#2 rows will be inserted above row number 1
```

insertTableRowBelow

Use insertTableRowBelow to insert row(s) in a table below a specific row.

Return Type

VOID

Syntax

```
insertTableRowBelow <rowNumber> <numberofRows>
```

Parameters

Parameter	Type	Description
`rowNumber`	`INT`	The row number below which new row(s) are to be added. This parameter is required.
`numberofRows`	`INT`	The number of rows to be inserted. This parameter is required.

Examples

```
insertTableRowBelow 5 1
#1 row will be inserted just below row number 5
```

lockAspectRatio

Locks the height and width aspect ratio of the selected image when it is resized.

Return Type

NONE

Syntax

```
lockAspectRatio <option>
```

Parameters

Parameter	Type	Description
`options`	`STRING`	whether to lock aspectRatio ratio This parameter is required.

Examples

```
lockAspectRatio ON
```

lockTableColumns

Restricts the number of columns in a table. If set to true, the number of columns cannot be changed, and if set to false, the number of columns can be changed.
Note: This command will not work on non-editable tables.

Return Type

NULL

Syntax

```
lockTableColumns <isLockable>
```

Parameters

Parameter	Type	Description
`isLockable`	`BOOL`	Specifies if the number of columns can be changed. If set to false, the number of columns cannot be changed and if true, the number of columns can be changed. This parameter is required.

Examples

```
#To restrict the number of columns
lockTableColumns true

#To remove restrictions on the number of columns
lockTableColumns false
```

Related Commands

lockTableRows

lockTableRows

Restricts the number rows in a table. If set to true, the number of rows cannot be changed, and if set to false, the number of rows can be changed.
Note: This command will not work on non-editable tables.

Return Type

NULL

Syntax

```
lockTableRows <isLockable>
```

Parameters

Parameter	Type	Description
`isLockable`	`BOOL`	Specifies if the number of rows can be changed. If set to false, the number of rows cannot be changed and if true, the number of rows can be changed. This parameter is required.

Examples

```
#To restrict the number of rows
lockTableRows true

#To remove restrictions on the number of rows
lockTableRows false
```

Related Commands

lockTableColumns

makeNetAsBaseNet

It identifies the selected net as the winning net. It needs to have a net selected on canvas to work. This command comes in use mostly between aliased nets in which there is one winning net among all the aliased nets. To make any other net as the base net, this command is run after selecting the candidate net on the canvas. Then after running this command for the selected net, that net becomes the winning net.
The physical net name of the winning net is passed to Allegro PCB Editor.

Return Type

BOOL

Syntax

```
makeNetAsBaseNet
```

Examples

```
makeNetAsBaseNet
```

mergeTableCells

Merges the cell range specified for the currently selected table. Contents of all the cells get deleted and only the content of the first cell, indicated by the startRow and startColumn, remain in the merged cell.

This command does not work if multiple tables are selected.

Return Type

NULL

Syntax

```
mergeTableCells <startRow> <startColumn> <endRow> <endColumn>
```

Parameters

Parameter	Type	Description
`startRow`	`INT`	The row number from where the merging of cells starts This parameter is required.
`startColumn`	`INT`	The column number from where the merging of cells starts This parameter is required.
`endRow`	`INT`	The row number until where the cells should merge This parameter is required.
`endColumn`	`INT`	The column number until where the cells should merge This parameter is required.

Examples

```
#Merges the second and third cells of the first column of a table:
mergeTableCells 2 1 3 1

#Merges the first row of a table with 4 columns
mergeTableCells 1 1 1 4
```

Related Commands

unmergeTableCells

unmergeAllTableCalls

openHyperlink

Opens the hyperlink associated with an item.

Return Type

NONE

Syntax

```
openHyperlink <link_address>
```

Parameters

Parameter	Type	Description
`link_address`	`STRING`	Specify a hyperlink address. Hyperlink address can be a web page, email address or schematic page This parameter is required.

Examples

```
openHyperlink {http://www.cadence.com}
```

openNPR

This Tcl command is used to launch either NPR form or NPR dashboard tab in System Capture, when Pulse is running in ad-hoc mode.
The launch of either of above mentioned tabs depends upon the bool value of first parameter. If false, NPR form will be launched else, NPR dashboard.

Return Type

None

Syntax

```
cpunicorn::openNPR <isDashboard> <additionalParam>
```

Parameters

Parameter	Type	Description
`isDashboard`	`bool`	Determines whether the NPR form or NPR dashboard is launched. This parameter is required.
`additionalParam`	`const char*`	Appends any additional parameters to the URL, which is loaded to launch the NPR form or dashboard. If no additional parameter is required, an empty value can be used. This parameter is required.

Examples

```
cpunicorn::openNPR true ""
cpunicorn::openNPR false ""
```

reassignBlockRefdes

Return Type

NONE

Syntax

```
sch::reassignBlockRefdes
```

reassignRefdes

This command assigns reference designators to selected instances based on current Packaging options, assigned reference designator pattern in the project and placement of instance on page and graphical location of the instances.

Return Type

NONE

Syntax

```
reassignRefdes ?-insts <list_inst_spaths>? | ?-block <block_name>? ?-pages
<list_pages>? ?-flat_only?
```

Parameters

Parameter	Type	Description
`list_inst_spaths`	`LIST`	Used with -insts. Specify list of spaths for the instances for which reference designator needs to be re-assigned This parameter is optional.
`block_name`	`STRING`	Used with -block. Specify block name for which instance reference designators need to be re-assigned This parameter is optional.
`list_pages`	`LIST`	Used with -pages. Specify list of page numbers of <block_name> whose instances need to re-assigned reference designators This parameter is optional.
`flat_only`	`BOOLEAN`	When this option is used, the instances of hierarchical block instances in the current block being processed are not used for reassignment This parameter is optional.

Examples

```
#reassign reference designator for selected instances on schematic canvas
selectObject -occPath @worklib.low(tbl_1):page(1) -type INST + 11906 10966
selectObject -occPath @worklib.low(tbl_1):page(1) -type INST + 14839 9214
selectObject -occPath @worklib.low(tbl_1):page(1) -type INST + 11407 9214
reassignRefdes
#reassign reference designator for a block and hierarchy below
reassignRefdes -block memory
#reassign reference designator for a set of pages in a block
reassignRefdes -block memory -pages [list 1,3,4]
#reassign reference designator for a block and skip reassigning for instances of block
instances in the current block
reassignRefdes -block memory -flatonly
```

Related Commands

reassignBlockRefdes

removeBends

Removes bends which are not required in the connector element.

Return Type

INT

Syntax

```
removeBends
```

Examples

```
removeBends
```

removeHyperlink

Use removeHyperlink to remove the hyperlink associated with an item.

Return Type

NONE

Syntax

```
removeHyperlink -type <item_type> ?-cell <row_no>,<column_no>? ?-posStart
<start_position> -posEnd <end_position> | -n <property_name> | -posStart
<start_position> -posEnd <end_position>?
```

Parameters

Parameter	Type	Description
`item_type`	`STRING`	Supported item types are RICH_NOTE,TABLE, and OCCURRENCE_PROP This parameter is required.
`row_no`	`INT`	The row number where the hyperlink is to be added. Applicable if item_type is Table This parameter is optional.
`column_no`	`INT`	The column number where the hyperlink is to be added. Applicable if item_type is Table This parameter is optional.
`start_position`	`INT`	Position from where the hyperlink text starts. Start position should start from 0 and It goes to (number of character -1) in the item. Start position should be less than or equal to end position. Applicable if item_type is Table or RICH_NOTE. This parameter is optional.
`end_position`	`INT`	Position from where the hyperlink text ends. End position should start from 0 and It goes to (number of character -1) in the item. End position should be greater than or equal to start position. Applicable if item_type is Table or RICH_NOTE. This parameter is optional.
`property_name`	`STRING`	Specify the name of occurrence property on which hyperlink needs to be added. Applicable if item_type is OCCURANCE_PROP. This parameter is optional.

Examples

```
#removes hyperlinks on note start from pos 10 to 13.
removeHyperlink -type RICH_NOTE -posStart 10 -posEnd 13
#removes hyperlink on table start from pos 0 to 9.
removeHyperlink -type TABLE -cell {0,0} -posStart 0 -posEnd 9
#removes a hyperlinks on occurrence property
removeHyperlink -type OCCURRENCE_PROP -name {DSSAD} -t {swsw}
```

resize

Resizes the selected elements on the page. Only wire, block diagrams, and drawing items can be resized. Components and pins cannot be resized. The command must have two points, either top or bottom and either left or right, specifying a valid XY coordinate on the page.

To resize a wire, you have the following options:

Single wire resize to resize a single wire in any direction and route will follow the mouse path and generate a command accordingly. The command will have -s to denote single wire resize.

Multiple wire resize to resize multiple horizontal or vertical wires. In case of multiple wires, resize all the selected wires should be horizontal or vertical only. Both vertical wires and horizontal wires cannot be resized together.

L Point resize:
User can resize by holding L point of two wire segments. Only one L point can be resized in one operation.

Return Type

NONE

Syntax

```
resize ?-pg <spath>? ?[-s <list> | -m]? ?[- top <point>]? ?[-bottom <point>]? ?[-right
<point>]? ?[- left <point>]? ?[ -selPt<xpt, ypt>] ?
```

Parameters

Parameter	Type	Description
`-pg`	`STRING`	Name of the page which contains the element to be resized. This parameter is optional. Default value is `current page`.
`-value`	`INT`	Number of points by which the top edge of the block frame is moved. This value is negative when dragging the top edge towards the center and positive while moving away from the center. This parameter is optional.
`-bottom`	`INT`	Number of points by which the bottom edge of the block frame is moved. This value is negative when dragging the bottom edge towards the center and positive while moving away from the center. This parameter is optional.
`-left`	`INT`	Number of points by which the left edge of the block frame is moved. This value is negative when dragging the left edge towards the center and positive while moving away from the center. This parameter is optional.
`-right`	`INT`	Number of points by which the right edge of the block frame is moved. This value is negative when dragging the right edge towards the center and positive while moving away from the center. This parameter is optional.
`-selPt`	`POINT`	Specifies the coordinates of the selected route. This argument is required only for resizing wire segment. This parameter is optional.

Examples

```
#Single line resize
resize -pg @worklib.preoj1(tbl_1):Page(1) -s [list 7850,4400 11400,4400 11400,4400
11400,2800] -bottom -1600 -right -4150 -selPt 15550 4400

#Multiple horizontal lines resize
resize -pg @worklib.preoj1(tbl_1):Page(1) -m -bottom 0 -right 1050 -selPt 0 0

#Multiple vertical lines resize
resize -pg @worklib.preoj1(tbl_1):Page(1) -m -bottom 750 -right 0 -selPt 0 0

#Single wire L-point resize
resize -pg @worklib.preoj1(tbl_1):Page(1) -s [list ] -bottom -500 -right 250 -selPt
1750 4500

#Supported for non-wire objects
resize -pg @worklib.t1(tbl_1):page(1) -top -350 -left 0 -selPt 10700 4800
resize -pg @worklib.t1(tbl_1):page(1) -bottom 400 -right 750
resize -pg @worklib.t1(tbl_1):page(1) -bottom 480 -right 440
resize -pg @worklib.t1(tbl_1):page(1) -top -150 -left 1340
resize -pg @worklib.t1(tbl_1):page(1) -top -20 -right 670
```

Related Commands

addRectangle

addBlock

drawWire

selectObject

move

rotate

Rotates the currently selected block on the canvas to the right (clockwise) or to the left (counter-clockwise).

Return Type

BOOL

Syntax

```
rotate <RotationType> ?angle?
```

Parameters

Parameter	Type	Description
`RotationType`	`STRING`	Direction of rotation. This parameter is required.
`angle`	`INT`	Value of the rotation angle. This parameter is optional. Default value is `90`.

Examples

```
rotate right 90
```

sendBackward

Sends the selected element(s) one level backwards on the schematic page.

Return Type

INT

Syntax

```
sendBackward
```

Examples

```
sendBackward
```

sendToBack

Sends the selected element(s) to the back of all the elements on the schematic page.

Return Type

INT

Syntax

```
sendToBack
```

Examples

```
sendToBack
```

setAlternateFillColor

Use setAlternateFillColor to specify the color name or the hexadecimal value of the color to set as the secondary fill color.

Return Type

NONE

Syntax

```
setAlternateFillColor <clr>
```

Parameters

Parameter	Type	Description
`clr`	`COLOR`	Color name or a hexadecimal value for the color to set as the secondary fill color. This parameter is required.

Examples

```
#command to set the secondary fill color to transparent.
setAlternateFillColor none
#command to set the secondary fill color to yellow.
setAlternateFillColor yellow
```

setBackgroundColor

Use setBackgroundColor to set the background color of the schematic page.

Return Type

NONE

Syntax

```
setBackgroundColor <color_type>
```

Parameters

Parameter	Type	Description
`color_type`	`COLOR`	Specify color name or a hexadecimal value for the background color. To make the background transparent, specify the parameter value as 'none'. This parameter is required.

Examples

```
#command to set the background of the schematic page as transparent.
setBackgroundColor none
#command to set the background of the schematic page as black.
setBackgroundColor BLACK
```

setConnectorWidth

Changes the width of the connector.

Return Type

NONE

Syntax

```
setConnectorWidth <width_value>
```

Parameters

Parameter	Type	Description
`width_value`	`INT`	Width of the connector This parameter is required.

Examples

```
setConnectorWidth 20
```

setFill

It fills the selected shape with transparent color. In other words, it fills with no color.

Return Type

NONE

Syntax

```
setFill none
```

Examples

```
setFill none
It fills selected shape with no color.
```

Related Commands

setFillColor

setLineColor

setHeaderFillColor

setTextColor

setBackgroundColor

setHeaderFillColor

Fills the header row of the selected table object with the color. On failure, an empty list is returned.

Return Type

INT

Syntax

```
setHeaderFillColor <color>
```

Parameters

Parameter	Type	Description
`color`	`COLOR`	Fill color of the header row. Predefined color names or RGB values can be used This parameter is required.

Examples

```
#Command to select the table object at (1000, 600) coordinates on page(1) of
#labrun block and fills the header row with blue color
selectObject -occPath @worklib.labrun(tbl_1):page(1) -type TABLE 1000 600
setHeaderFillColor blue

#Command to select the table object at (1000, 600) coordinates on page(1) of
#labrun block and fills the header row with yellow color (RGB - #ffff00).
selectObject -occPath @worklib.labrun(tbl_1):page(1) -type TABLE 2000 600
setHeaderFillColor #ffff00
```

Related Commands

selectObject

setLineBeginStyle

Sets the begin/end style of block arrows. The following style names are supported:

ArrowDiamond
ArrowPointed
Bundle
Bus
DoubleBundleArrow
DoubleBusArrow
DoubleWireArrow
LeftBundleArrow
LeftBusArrow
LeftWireArrow
PCIeArrow
RightBundleArrow
RightBusArrow
RightWireArrow
Wire

Return Type

NONE

Syntax

```
setLineBeginStyle <styleName>
```

Parameters

Parameter	Type	Description
`styleName`	`STRING`	Begin/end style This parameter is required.

Examples

```
# sets a thin arrow shape on the left-end of selected block connector
setLineBeginStyle LeftWireArrow
```

Related Commands

setLineStyle

setLineWidth

setLineColor

setLineCap

Use setLineCap to change the open-end style of a shape to any of following parameters:
1. square-cap
2. flat-cap
3. round-cap

Return Type

NONE

Syntax

```
setLineCap <cap-style>
```

Parameters

Parameter	Type	Description
`cap-style`	`STRING`	Style of the shape This parameter is required.

Examples

```
#command to change open-end to square-cap
setLineCap square-cap
```

setLineColor

Use setLineColor to change the line color of the currently selected element(s) on the schematic page. In case the shape is selected, the border color of the shape changes.

Return Type

NONE

Syntax

```
setLineColor <colorcode>
```

Parameters

Parameter	Type	Description
`colorName`	`STRING`	Color name or the hexadecimal value for the color. This parameter is required.

Examples

```
#command to set the line color of the selected item to green
setLineColor #00FF00
#command to set the line color of the selected item to black
setLineColor black
```

setLineEndStyle

Use setLineEndStyle to change the arrow style on the block arrows.

Return Type

NONE

Syntax

```
setLineEndStyle <arrow-style>
```

Parameters

Parameter	Type	Description
`arrow-style`	`STRING`	Style for the arrow This parameter is required.

Examples

```
#command to change arrow style to DoubleWireArrow
setLineEndStyle DoubleWireArrow
```

setLineJoin

Use setLineJoin to change the style of the joints of the selected element on the schematic page.

Return Type

NONE

Syntax

```
setLineJoin <type>
```

Parameters

Parameter	Type	Description
`type`	`STRING`	Specifies the join type. Valid values are: miter-join, round-join, bevel-join This parameter is required.

Examples

```
setLineJoin miter-join
setLineJoin round-join
setLineJoin bevel-join
```

setLineType

Use setLineType to change the line style of the currently selected element(s) on the schematic page.

Style can be of the following types:
- solid
- dash
- dot
- dash-dot
- dash-dot-dot

Wires/Bus/NetGroup will have additional styles
- multi-core
- single-core
- twisted

Return Type

STRING

Syntax

```
setLineType <line_style>
```

Parameters

Parameter	Type	Description
`line_style`	`STRING`	line style. Supported values are: dash, dot, dash-dot, This parameter is required.

Examples

```
setLineType solid
setLineType dash
setLineType dot
setLineType dash-dot
setLineType dash-dot-dot
setLineType multi-core
setLineType single-core
setLineType ribbon
setLineType twisted
```

setLineWidth

Use setLineWidth to change the line width of the currently selected element(s) on the schematic page.

Return Type

NONE

Syntax

```
setLineWidth <value>
```

Parameters

Parameter	Type	Description
`value`	`INT`	Integer value to specify line width. Valid values are integers between 1 to 72. This parameter is required.

Examples

```
setLineWidth 10
```

setLock

Locks or unlocks the selected component instance.

Return Type

BOOL

Syntax

```
setLock -type <item_type> -state <locked_state>
```

Parameters

Parameter	Type	Description
`item_type`	`STRING`	The type of item being locked or unlocked. This parameter is required. Default value is `INST`.
`locked_state`	`BOOLEAN`	Indicates whether a component is locked or unlocked. This parameter is required.

Examples

```
setLock -type POWER -state ON
```

setOpacity

Sets the opacity of the selected image on the current page.

Return Type

BOOLEAN

Syntax

```
setOpacity <value>
```

Parameters

Parameter	Type	Description
`value`	`FLOAT`	opacity value of selected image This parameter is required.

Examples

```
setOpacity 0.31
```

setSubscript

Converts the selected text to a subscript or converts the selected subscript to regular text, depending on the parameter value.

Return Type

BOOLEAN

Syntax

```
setSubscript <choice> -posStart <value> -posEnd <value>
```

Parameters

Parameter	Type	Description
`choice`	`STRING`	If the parameter value is 'true', the selected regular text is converted to a subscript. When the parameter value is set to 'false', the command changes selected subscript to regular text. This parameter is required.
`posStart`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the start character of the text to be formatted. This parameter is optional.
`posEnd`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the last character of the text to be formatted. This parameter is optional.

Examples

```
#command to covert the text from 10th to 12th position of the selected
#string to subscript.
setSubscript true -posStart 10 -posEnd 12
```

Related Commands

setSuperscript

setSuperscript

It formats the selected text of the selected note at a specific position as a superscript of the regular text.

Return Type

BOOLEAN

Syntax

```
setSuperscript <choice> -posStart <value> -posEnd <value>
```

Parameters

Parameter	Type	Description
`choice`	`STRING`	When set to 'true', the command converts regular text as a superscript. If this parameter value is set to 'false', the command change the superscript to regular text. This parameter is required.
`posStart`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the start character of the text to be formatted. This parameter is optional.
`posEnd`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the last character of the text to be formatted. This parameter is optional.

Examples

```
setSuperscript true -posStart 10 -posEnd 11
```

Related Commands

setSubscript

setTableCellStyleable

Controls whether the style can be changed for cells. If the style changes are disabled, only the content can be changed. If changes to the table style are disabled, then the styles of the cells cannot be changed.

Return Type

NULL

Syntax

```
setTableCellStyleable <isStyleable> <startRow> <startColumn> <endRow> <endColumn>
```

Parameters

Parameter	Type	Description
`isStyleable`	`BOOL`	Specifies if the style for cells can be changed. If set to false, the style of the cells cannot be changed and if true, style can be changed. This parameter is required.
`startRow`	`INT`	Row number from where style changes for multiple cells starts This parameter is required.
`startColumn`	`INT`	Column number from where style changes for multiple cells starts This parameter is required.
`endRow`	`INT`	Row number where style changes for multiple cells ends This parameter is required.
`endColumn`	`INT`	Column number where style changes for multiple cells ends This parameter is required.

Examples

```
#Command to disable style changes for multiple cells from 1,1 to 4,4
setTableCellStyleable false 1 1 4 4
```

Related Commands

setTableStyleable

setTableRowStyleable

setTableColumnStyleable

setTableColumnStyleable

Controls whether the column style can be changed. If the style changes are disabled, only the content can be changed. If changes to the table style are disabled, then the styles of the columns cannot be changed.

Return Type

NULL

Syntax

```
setTableColumnStyleable <isStyleable> <column_number>
```

Parameters

Parameter	Type	Description
`isStyleable`	`BOOL`	Specifies if the style for a column can be changed. If set to false, the column style cannot be changed and if true, row style can be changed. This parameter is required.
`column_number`	`INT`	Column number whose style permission are being changed This parameter is required.

Examples

```
#Command to disable the style changes to the third column
setTableColumnStyleable false 3
```

Related Commands

setTableStyleable

setTableRowStyleable

setTableCellsStyleable

setTableProperty

It adds a property to the selected table object on the schematic page. The property can be used to identify table objects in Tcl scripts. It returns 0 on success.

Return Type

INTEGER

Syntax

```
setTableProperty <prop_name> <prop_value>
```

Parameters

Parameter	Type	Description
`prop_name`	`STRING`	Specifies property name to a table This parameter is required.
`prop_value`	`STRING`	Specifies property value to a table This parameter is required.

Examples

```
#set a property "table_type" with value "power_domains" on a table object listing the
power domains used in the design
setTableProperty "table_type" "power_domains"
#get the property for subsequent use
sch::dbGetCSPropNameVal [sch::dbGetSelectedItems [sch::dbGetActivePage]]
```

Related Commands

dbGetActivePage

dbGetSelectedItems

dbGetCSPropNameVal

setTableRowAsHeader

Set a row as the table header. Any row can be made the header row depending on the following:
1. If the row above the specified row is a header row.
2. If the specified row is the last row of the table

The command will not work if
1. The specified row is already a header row
2. If it does not follow the conditions mentioned above.

Return Type

NULL

Syntax

```
setTableRowAsHeader <isMarkable> <row_number>
```

Parameters

Parameter	Type	Description
`isMarkable`	`BOOL`	Specifies if a header row is being assigned. If set to true, then the specified row is made the table header and if set to, no header row can be specified. This parameter is required.
`row_number`	`INT`	The row number to be made the table header This parameter is required.

Examples

```
#To specify a row of a table as its header
setTableRowAsHeader true 1
```

setTableRowStyleable

Controls whether the row style can be changed. If the style changes are disabled, only the content can be changed. If changes to the table style are disabled, then the styles of the cells cannot be changed.

Return Type

NULL

Syntax

```
setTableRowStyleable <isStyleable> <row_number>
```

Parameters

Parameter	Type	Description
`isStyleable`	`BOOL`	Specifies if the style for a row can be changed. If set to false, the row style cannot be changed and if true, row style can be changed. This parameter is required.
`row_number`	`INT`	Row number whose style permission are being changed This parameter is required.

Examples

```
#To disable the style changes of the table header
setTableRowStyleable false 1
```

Related Commands

setTableStyleable

setTableColumnStyleable

setTableCellsStyleable

setTableStyleable

Controls whether the table style can be changed. If the style changes are disabled, only the content can be changed.

Return Type

NULL

Syntax

```
setTableStyleable <isStyleable>
```

Parameters

Parameter	Type	Description
`isStyleable`	`BOOL`	Specifies if the table style can be changed. If set to false, table style cannot be changed and if true, table style can be changed. This parameter is required.

Examples

```
#Command to allow style changes of a table
setTableStyleable true

#Command to disable style changes of a table
setTableStyleable true
```

Related Commands

setTableRowStyleable

setTableColumnStyleable

setTableCellsStyleable

setTextBold

It formats the selected regular text as bold or bold text as regular text.

Return Type

BOOLEAN

Syntax

```
setTextBold <choice> -posStart <value> -posEnd <value>
```

Parameters

Parameter	Type	Description
`choice`	`STRING`	When set to 'true', the command converts regular text to bold. If this parameter value is set to 'false', the command changes bold text to regular text. This parameter is required.
`posStart`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the start character of the text to be formatted. This parameter is optional.
`posEnd`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the last character of the text to be formatted. This parameter is optional.

Examples

```
setTextBold true -posStart 6 -posEnd 29
```

setTextColor

It changes the color of the selected text to the specified value.

Return Type

BOOLEAN

Syntax

```
setTextColor <color> -posStart <value> -posEnd <value>
```

Parameters

Parameter	Type	Description
`color`	`STRING`	Color name or the hexadecimal value of the color. This parameter is required.
`posStart`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the start character of the text to be formatted. This parameter is optional.
`posEnd`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the last character of the text to be formatted. This parameter is optional.

Examples

```
setTextColor #00a2e8 -posStart 26 -posEnd 26
```

Related Commands

setTextBold

setTextFont

setTextItalic

setTextSize

setTextUnderline

setTextWordWrap

setTextFont

It sets the font of the selected text to the specified value.

Return Type

BOOLEAN

Syntax

```
setTextFont <font_name> ?-posStart <value>? ?-posEnd <value>?
```

Parameters

Parameter	Type	Description
`font_name`	`STRING`	Name of font This parameter is required.
`-posStart`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the start character of the text to be formatted. This parameter is optional.
`-posEnd`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the last character of the text to be formatted. This parameter is optional.

Examples

```
#System Capture is not in the Text Edit Mode, and
#the text to be modified is selected.
setTextFont Arial
#System Capture is in the Text Edit Mode. The text to be modified
#is not selected, but the cursor is position on the string to be modified.
setTextFont Arial -posStart 5 -posEnd 6
```

Related Commands

setTextBold

setTextColor

setTextItalic

setTextSize

setTextUnderline

setTextWordWrap

setTextItalic

It formats the selected regular text at a specific position as italicized or italicized text as regular text.

Return Type

BOOLEAN

Syntax

```
setTextItalic <choice> ?-posStart <value>? ?-posEnd <value>?
```

Parameters

Parameter	Type	Description
`choice`	`STRING`	Use "true" to format regular text as italicized or "false" to change italicized text to regular text. This parameter is required.
`posStart`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the first character of the text to be formatted. This parameter is optional.
`posEnd`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the last character of the text to be formatted. This parameter is optional.

Examples

```
setTextItalic true -posStart 31 -posEnd 51
```

Related Commands

setTextBold

setTextColor

setTextFont

setTextSize

setTextUnderline

setTextWordWrap

setTextSize

It changes the size of the text of the selected note(s).

Return Type

BOOLEAN

Syntax

```
setTextSize <value> ?-posStart <value> -posEnd <value>?
```

Parameters

Parameter	Type	Description
`value`	`INT`	Specifies the text size. This parameter is required.
`posStart`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the start character of the text to be formatted. This parameter is optional.
`posEnd`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the start character of the text to be formatted. This parameter is optional.

Examples

```
setTextSize 29
setTextSize 29 -posStart 6 -posEnd 7
```

Related Commands

setTextBold

setTextColor

setTextFont

setTextItalic

setTextUnderline

setTextWordWrap

setTextUnderline

It underlines the selected text at a specific position or removes the underline from the text.

Return Type

BOOLEAN

Syntax

```
setTextUnderline <choice> ?-posStart<value>? ?-posEnd<value>?
```

Parameters

Parameter	Type	Description
`choice`	`STRING`	Use "true" to underline regular text or "false" to remove the underline. This parameter is required.
`posStart`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the first character of the text to be formatted. This parameter is optional.
`posEnd`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the last character of the text to be formatted. This parameter is optional.

Examples

```
setTextUnderline true -posStart 31 -posEnd 51
```

Related Commands

setTextBold

setTextColor

setTextFont

setTextItalic

setTextSize

setTextWordWrap

setTextWordWrap

It wraps the selected text to the next line to fit it in the available width of the containing box.

Return Type

BOOLEAN

Syntax

```
setTextWordWrap <choice> ?-posStart <value>? ?-posEnd <value>?
```

Parameters

Parameter	Type	Description
`choice`	`STRING`	Use "true" to enable or "false" to disable word wrapping. This parameter is required.
`posStart`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the first character of the text to be formatted. This parameter is optional.
`posEnd`	`INT`	This parameter value is read only when System Capture is in the textEdit mode. This parameter specifies the position of the last character of the text to be formatted. This parameter is optional.

Examples

```
setTextWordWrap true -posStart 32 -posEnd 59
```

Related Commands

setTextBold

setTextColor

setTextFont

setTextItalic

setTextSize

setTextUnderline

tableClearContents

It clears the contents of the table starting from the top-left cell to the bottom-right cell.

Return Type

NONE

Syntax

```
tableClearContents <topLeftRow> <topleftCol> <bottomRightRow> <BottomRightCol>
```

Parameters

Parameter	Type	Description
`topLeftRow`	`INT`	The top-left row number. Row number starts from 1 and goes up to no of rows of a table This parameter is required.
`topleftCol`	`INT`	The top-left column number. Column number starts from 1 and goes up to no of columns of a table This parameter is required.
`bottomRightRow`	`INT`	The bottom-right row number. Row number starts from 1 and goes up to no of rows of a table This parameter is required.
`BottomRightCol`	`INT`	The bottom-right column number. Column number starts from 1 and goes up to no of columns of a table This parameter is required.

Examples

```
tableClearContents 1 2 2 2
```

tableColumnResize

It resizes the column width of a table.

Return Type

NONE

Syntax

```
tableColumnResize <columnNumber> <width>
```

Parameters

Parameter	Type	Description
`columnNumber`	`INT`	Specify column number of a table whose width needs to be modified. Column number should start from 1 and goes up to no of columns in the table. This parameter is required.
`width`	`INT`	Specify the width of the column. This parameter is required.

Examples

```
Modifies the width of column number 2.
tableColumnResize 2 400
```

Related Commands

tableRowResize

tableFitRowToContent

Resizes a row of the table to fit the contents of the table. Returns 0 for successful operation.

Return Type

INT

Syntax

```
tableFitRowToContent <rowNumber>
```

Parameters

Parameter	Type	Description
`rowNumber`	`INT`	Number of rows. This parameter is required.

Examples

```
tableFitRowToContent 3
```

tableFitToContent

It resizes a table to fit the contents of the table. It will remove extra space from the table and the table will be fit to its contents.

Return Type

NONE

Syntax

```
tableFitToContent
```

Related Commands

tableFitRowToContent

tablePlainTextEdit

It modifies the text in a specified cell of the selected table.

Return Type

NONE

Syntax

```
tablePlainTextEdit <row_number> <column_number> <plain_text>
```

Parameters

Parameter	Type	Description
`row_number`	`INT`	Specify row number of a table whose text needs to be edited. Row number should start from 1 and goes up to no of rows in the table. This parameter is required.
`column_number`	`INT`	Specify column number of a table whose text needs to be edited. Column number should start from 1 and goes up to no of columns in the table. This parameter is required.
`plain_text`	`STRING`	Specify a plain text. This parameter is required.

Examples

```
#a text "Hello there" will be set on cell 1,1 of a table.
tablePlainTextEdit 1 1 {Hello there}
```

Related Commands

textRichTextEdit

tableRichTextEdit

It modifies the text in a specified cell of the selected table. It supports text formatting, such as bold, italics, and underlining, as well as different fonts, font sizes, and colored text.

Return Type

NONE

Syntax

```
tableRichTextEdit <row_number> <column_number> <richtext>
```

Parameters

Parameter	Type	Description
`row_number`	`INT`	Specify row number of a table whose text needs to be edited. Row number should start from 1 and goes up to no of rows in the table. This parameter is required.
`column_number`	`INT`	Specify column number of a table whose text needs to be edited. Column number should start from 1 and goes up to no of columns in the table. This parameter is required.
`richtext`	`STRING`	Specify a rich text. This parameter is required.

Examples

```
#the text "Hello there" with bold characters will be set
#on cell 1,1 of a table.
tableRichTextEdit 1 1 {&lt;b&gt;Hello there&lt;b&gt;}
```

Related Commands

tablePlainTextEdit

tableRowResize

It resizes the row height of a table.

Return Type

NONE

Syntax

```
tableRowResize <rowNumber> <height>
```

Parameters

Parameter	Type	Description
`rowNumber`	`INT`	Specify row number of a table whose height needs to be modified. Row number should start from 1 and goes up to no of rows in the table. This parameter is required.
`height`	`INT`	Specify the height of the column. This parameter is required.

Examples

```
#command to modify the height of row number 1.
tableRowResize 1 300
```

textAlign

Changes the alignment of the text.

Return Type

NONE

Syntax

```
textAlign <value> ?-posStart <value>? ?-posEnd <value>?
```

Parameters

Parameter	Type	Description
`value`	`STRING`	Type of alignment. This parameter is required.
`-posStart <value>`	`INT`	Position of the start character of the text to be aligned. Used in the textEdit mode only. This parameter is optional.
`-posEnd <value>`	`INT`	Position of the end character of the text to be aligned. Used in the textEdit mode only. This parameter is optional.

Examples

```
textAlign left -posStart 55 -posEnd 67
```

textEdit

Edits text at the specified location on the schematic page.

Return Type

NONE

Syntax

```
textEdit ?-pg <page_name>? -s <new_text> -pos [list <x-coordinate> <y-coordinate>]
```

Parameters

Parameter	Type	Description
`-pg <page_name>`	`STRING`	Name of the page where the text has to be edited. This parameter is optional. Default value is `current page`.
`-s <new_Text>`	`STRING`	The new text string. This parameter is required.
`-pos [list <x-coordinate> <y-coordinate>]`	`INTEGER`	The XY-coordinates of the top-left corner of the grid. This parameter is required.

Examples

```
textEdit -pg page(1) -pos [list 6997 11550] -s a
```

ungroup

Ungroups the elements of an existing group.

Return Type

NONE

Syntax

```
ungroup
```

Examples

```
ungroup
```

unhilightObject

Moves the focus away from the highlighted object on the schematic page.

Return Type

NONE

Syntax

```
unhilightObject ?[<top_x-coordinate> <top_y-coordinate> <bottom_x-coordinate>
<bottom_y-coordinate>]? ?[mode]? [-spath <path>] [<page_path>]
```

Parameters

Parameter	Type	Description
`[<top_x-coordinate> <top_y-coordinate> <bottom_x-coordinate> <bottom_y-coordinate>]`	`INT`	Grid coordinates of the component. This parameter is optional.
`[mode]`	`STRING`	The selection mode. This parameter is optional.
`[-sPath <path>]`	`STRING`	The sPath name This parameter is required.
`[<page_path>]`	`STRING`	The page path. This parameter is required.

Examples

```
unhighlightObject -sPath @worklib.a(tbl_1):\\i96\\
```

unmergeAllTableCells

Unmerges all merged cells in the selected table. This command does not work if multiple tables are selected.

Return Type

Void

Syntax

```
unmergeAllTableCells
```

Examples

```
# To merge table cells
mergeTableCells 2 1 3 1
mergeTableCells 4 4 5 5

# To unmerge all the merged cells in a table
unmergeAllTableCells
```

Related Commands

mergeTableCells

unmergeTableCells

unmergeTableCells

Unmerges the merged cells of the currently selected smart table. All merged cells present between the start and end range get separated. The content of each merged cell group gets copied to its starting row, column.

This command will not work if multiple tables are selected.

Return Type

NULL

Syntax

```
unmergeTableCells <startRow> <startColumn> <endRow> <endColumn>
```

Parameters

Parameter	Type	Description
`startRow`	`INT`	The row number from where the unmerging of cells starts This parameter is required.
`startColumn`	`INT`	The column number from where the unmerging of cells starts This parameter is required.
`endRow`	`INT`	The row number until where the cells need to unmerge This parameter is required.
`endColumn`	`INT`	The column number until where the cells need to unmerge This parameter is required.

Examples

```
#Command to unmerge the second cell of the first column
unmergeTableCells 2 1 2 1
```

Related Commands

mergeTableCells

unmergeAllTableCells

6

Physical View

Physical view of the schematic design gives its flattened view. That is, how the design will appear when taken to the board. Its major constituents are: physical nets, physical part definitions, physical part instances.Constituents of physical view are:

- **Physical part definition**

It represents the part definition in the physical design view.

- **Physical part instance**

It represents a part instance in the physical design view.

- **Physical Net**

It represents a net in the physical design view.

- **Physical Pin Instance**

It represents a pin in the physical design view.

- **Physical Function Instance**

It represents the functions in the physical design view.

- **Physical Part Definition Iterator**

Iterator for enumerating the physical part definitions in the design.

- **Physical Part Instance Iterator**

Iterator for enumerating the physical part instances in the design.

- **Physical Pin Iterator**

Iterator for enumerating the physical pins in the design.

- **Physical Net Iterator**

Iterator for enumerating the physical nets in the design.

- **Physical function Iterator**

Iterator for enumerating the function instances in the design.Also refer to general guidelines in logical view section for writing TCL script with logical and physical view commands.

createIPhysNetIter

Returns an iterator representing the beginning of the physical nets in the design.

The parameter, vSysNets can have one of the following values:
1 - system nets (internal nets) are to be included in the iteration list.
0 - system nets will not be iterated

Return Type

Physical Net Iterator

Syntax

```
createIPhysNetIter <ptr> <vSysNets>
```

Parameters

Parameter	Type	Description
`ptr`	`Design Object`	The design object for which the iterator is to be created. This parameter is required.
`vSysNets`	`BOOL`	A boolean value indicating whether the system will be included in the iterator list or not. This parameter is required.

Examples

```
createIPhysNetIter $ptr $vSysNets
```

Related Commands

getServer

IServer_findDesign

createPhysPartDefnIter

Returns an iterator representing the start of the physical part definition object.

The parameter, vbool, can have one of the following values:
1 - associated components are also included in the iteration list
0 - associated components will not be iterated using this iterator

Return Type

Physical Part Definition Iterator

Syntax

```
createPhysPartDefnIter <ptr> <vbool>
```

Parameters

Parameter	Type	Description
`ptr`	`Design Object`	The design object for which the iterator is to be created. This parameter is required.
`vbool`	`BOOL`	Specifies whether the associated components will be included in the iteration list. This parameter is required.

Examples

```
createPhysPartDefnIter $ptr $vbool
```

Related Commands

getServer

lServer_findDesign

get

Returns an object represented by the iterator in its current state.

Caution:
If the iterator is in its end state, the access operator can fail and lead to the termination of the program. Ensure that the iterator is not in its end state before accessing the object. The end state can be checked using the more method.

Return Type

Object

Syntax

```
get
```

Examples

```
set physPin [$physPinIter get]
# The get command being used on the physical pin iterator, physPinIter, returns the
object represented by the iterator.
# returned value is stored in the physPin variable.
```

Related Commands

IPhysPartDefn_beginPartInst

IPhysNet_beginPin

IPhysFunc_beginPin

createPhysPartDefnIter

createIPhysNetIter

getPhysNet

Returns the physical net object for the specified physical net.

Return Type

Physical Net

Syntax

```
getPhysNet <pDesignCxt> <PhysicalNetName>
```

Parameters

Parameter	Type	Description
`pDesignCxt`	`Design Context`	The design context object representing the design containing the specified physical net. This parameter is required.
`PhysicalNetName`	`String`	The string representing the physical net name. This parameter is required.

Examples

```
getPhysNet $pDesignCxt $PhysicalNetName
```

Related Commands

getServer

IServer_findDesign

getPhysPartInst

Returns the physical part instance object for the specified design instance.

Return Type

Physical Part Instance

Syntax

```
getPhysPartInst <pDesignContext> <refdes>
```

Parameters

Parameter	Type	Description
`pDesignContext`	`Design Context`	The design context object representing the design containing the specified instance. This parameter is required.
`refdes`	`String`	The string representing the reference designator for the part instance. This parameter is required.

Examples

```
getPhysPartInst $pDesignContext $refdes
```

Related Commands

getServer

IServer_findDesign

getPhysPinInst

Returns the physical net instance object for the specified instance terminal bit context.

Return Type

Physical Pin Instance

Syntax

```
getPhysPinInst <pDesignContext> <pInstTermBitContext>
```

Parameters

Parameter	Type	Description
`pDesignContext`	`Design Context`	The design context object representing the design containing the specified physical net. This parameter is required.
`pInstTermBitContext`	`Instance Terminal Bit Context`	The pointer to the instance terminal bit context object representing the design containing the specified physical net. This parameter is required.

Examples

```
getPhysPinInst $pDesignContext $pInstTermBitContext
```

Related Commands

getServer

IServer_findDesign

getPropName

The getPropName method is used to get the name of the current physical property on the object represented by the iterator. It returns the name of the physical property as string.

Return Type

Property Name

Syntax

```
getPropName
```

Examples

```
set propName [$propIter getPropName]
# The getPropName command used in the above example returns the name of the property
represented by the property iterator, propIter.
```

Related Commands

IPhysPartInst_beginProp

IPhysPartDefn_beginProp

getPropValue

The getPropValue method is used to get the value assigned to the current physical property on the object represented by the iterator. Returns a value of the physical property as string.

Return Type

Property Value

Syntax

```
getPropValue
```

Examples

```
set propValue [$propIter getPropValue]
# The getPropValue command used in the above example returns the value of the property
represented by the property iterator, propIter.
```

Related Commands

IPhysPartInst_beginProp

IPhysPartDefn_beginProp

increment

Works like an increment operator and increments the iterator value by 1. The incremented value of the iterator is stored in the iterator itself. Returns the iterator with its value incremented by 1.

Return Type

Object Iterator

Syntax

```
increment
```

Examples

```
set physPartInstIter [$physPartInstIter increment]
# In the above example, the value of the physical part instance iterator,
physPartInstIter is incremented by 1, so as to iterate over the next instance.
```

Related Commands

IPhysPartDefn_beginPartInst

IPhysNet_beginPin

IPhysFunc_beginPin

createPhysPartDefnIter

createIPhysNetIter

IPhysFunc_beginPin

Returns an iterator representing the beginning of pin enumeration for this function.

Return Type

Physical Pin Iterator

Syntax

```
IPhysFunc_beginPin <PhysFunc>
```

Parameters

Parameter	Type	Description
`PhysFunc`	`Physical Function Instance`	Specifies the object representing the function in the physical design view. This parameter is required.

Examples

```
IPhysFunc_beginPin $PhysFunc
```

Related Commands

getServer

IServer_findDesign

IPhysPartInst_beginFunc

IPhysFunc_findProp

Finds the specified property name within the specified function in the physical design.

Return Type

Property Value

Syntax

```
IPhysFunc_findProp <PhysFunc> <propName> <p>
```

Parameters

Parameter	Type	Description
`PhysFunc`	`Physical Function Instance`	Specifies the object representing the function in the physical design view. This parameter is required.
`propName`	`String`	The name of the property to be found. This parameter is required.
`p`	`BOOL`	A True value indicates that inherited properties should also be searched to find the specified property This parameter is required.

Examples

```
IPhysFunc_findProp $PhysFunc $propName $p
```

Related Commands

getServer

IServer_findDesign

IPhysPartInst_beginFunc

IPhysFunc_isSplit

Checks whether the function is a split function. Returns True if the function is a split function, else returns False.

Return Type

INT

Syntax

```
IPhysFunc_isSplit <PhysFunc>
```

Parameters

Parameter	Type	Description
`PhysFunc`	`Physical Function Instance`	Specifies the object representing the function in the physical design view. This parameter is required.

Examples

```
IPhysFunc_isSplit $PhysFunc
```

Related Commands

getServer

IServer_findDesign

IPhysPartInst_beginFunc

IPhysFunc_sectionNum

Finds the section number for the specified function. Returns an integer representing the section number.

Return Type

Section Number

Syntax

```
IPhysFunc_sectionNum <PhysFunc>
```

Parameters

Parameter	Type	Description
`PhysFunc`	`Physical Function Instance`	Specifies the object representing the function in the physical design view. This parameter is required.

Examples

```
IPhysFunc_sectionNum $PhysFunc
```

Related Commands

getServer

IServer_findDesign

IPhysPartInst_beginFunc

IPhysFunc_spath

Returns the system canonical path for the specified function.

Return Type

Canonical Path

Syntax

```
IPhysFunc_spath <PhysFunc>
```

Parameters

Parameter	Type	Description
`PhysFunc`	`Physical Function Instance`	Specifies the object representing the function in the physical design view. This parameter is required.

Examples

```
IPhysFunc_spath $PhysFunc
```

Related Commands

getServer

IServer_findDesign

IPhysPartInst_beginFunc

IPhysNet_beginPin

Returns an iterator representing the physical pins connected by the specified physical net.

Return Type

Physical Pin Iterator

Syntax

```
IPhysNet_beginPin <PhysNet>
```

Parameters

Parameter	Type	Description
`PhysNet`	`Physical Net`	Specifies the object representing the physical net in the design. This parameter is required.

Examples

```
IPhysNet_beginPin $PhysNet
```

Related Commands

getServer

IServer_findDesign

getPhysNet

IPhysNet_getLogicalNet

Use this method to get the logical net name assigned to the net. Returns the logical net for the physical net represented by this physical net object.

Return Type

Logical Net

Syntax

```
IPhysNet_getLogicalNet <PhysNet>
```

Parameters

Parameter	Type	Description
`PhysNet`	`Physical Net`	It specifies the object representing the physical net in the design This parameter is required.

Examples

```
IPhysNet_getLogicalNet $PhysNet
```

Related Commands

getServer

IServer_findDesign

getPhysNet

IPhysNet_name

Returns the physical net name.

Return Type

Physical Net Name

Syntax

```
IPhysNet_name <PhysNet>
```

Parameters

Parameter	Type	Description
`PhysNet`	`Physical Net`	Specifies the object representing the physical net in the design. This parameter is required.

Examples

```
IPhysNet_name $PhysNet
```

Related Commands

getServer

IServer_findDesign

getPhysNet

IPhysNet_scope

Returns a value representing the scope of the specified physical net.

The following values correspond to a specific net scope:
0 - local net
1 - interface net
2 - global net

Return Type

Scope

Syntax

```
IPhysNet_scope <PhysNet>
```

Parameters

Parameter	Type	Description
`PhysNet`	`Physical Net`	Specifies the object representing the physical net in the design. This parameter is required.

Examples

```
IPhysNet_scope $PhysNet
```

Related Commands

getServer

IServer_findDesign

getPhysNet

IPhysNet_spath

Returns the system canonical path for the specified physical net.

Return Type

Canonical Path

Syntax

```
IPhysNet_spath <PhysNet>
```

Parameters

Parameter	Type	Description
`PhysNet`	`Physical Net`	Specifies the object representing the physical net in the design. This parameter is required.

Examples

```
IPhysNet_spath $PhysNet
```

Related Commands

getServer

IServer_findDesign

getPhysNet

IPhysPartDefn_beginPartInst

Returns a part instance iterator representing the beginning of the part instances for this part definition.

Return Type

Part Instance Iterator

Syntax

```
IPhysPartDefn_beginPartInst <PhysPartDefn>
```

Parameters

Parameter	Type	Description
`PhysPartDefn`	`Physical Part Definition`	Specifies the object representing the physical part definition in the design This parameter is required.

Examples

```
IPhysPartDefn_beginPartInst $PhysPartDefn
```

Related Commands

getServer

IServer_findDesign

createPhysPartDefnIter

IPhysPartDefn_beginProp

Returns a physical part definition property iterator representing the beginning of properties for the part definition. Property name and value are retrieved from the iterator: getPropName, getPropValue

Return Type

Property Iterator

Syntax

```
IPhysPartDefn_beginProp <PhysPartDefn>
```

Parameters

Parameter	Type	Description
`PhysPartDefn`	`Physical Part Definition`	Specifies the object representing the physical part definition in the design. This parameter is required.

Examples

```
IPhysPartDefn_beginProp $PhysPartDefn
```

Related Commands

getServer

IServer_findDesign

createPhysPartDefnIter

getPropName

getPropValue

IPhysPartDefn_findProp

Returns the value of the specified property as a string in the part definition.

Return Type

Property Value

Syntax

```
IPhysPartDefn_findProp <PhysPartDefn> <strPropName>
```

Parameters

Parameter	Type	Description
`PhysPartDefn`	`Physical Part Definition`	Specifies the object representing the physical part definition in the design. This parameter is required.
`strPropName`	`String`	String representing the name of the property to be found. This parameter is required.

Examples

```
IPhysPartDefn_findProp $PhysPartDefn $strPropName
```

Related Commands

getServer

IServer_findDesign

createPhysPartDefnIter

IPhysPartDefn_name

Returns the name of the part definition object.

Return Type

Physical Part Definition Name

Syntax

```
IPhysPartDefn_name <PhysPartDefn>
```

Parameters

Parameter	Type	Description
`PhysPartDefn`	`Physical Part Definition`	Specifies the object representing the physical part definition in the design. This parameter is required.

Examples

```
IPhysPartDefn_name $PhysPartDefn
```

Related Commands

getServer

IServer_findDesign

createPhysPartDefnIter

IPhysPartDefn_numOfFunctions

Returns the number of functions in the specified part definition.

Return Type

Number of Functions

Syntax

```
IPhysPartDefn_numOfFunctions <PhysPartDefn>
```

Parameters

Parameter	Type	Description
`PhysPartDefn`	`Physical Part Definition`	Specifies the object representing the physical part definition in the design. This parameter is required.

Examples

```
IPhysPartDefn_numOfFunctions $PhysPartDefn
```

Related Commands

getServer

IServer_findDesign

createPhysPartDefnIter

IPhysPartInst_beginFunc

Returns an iterator representing the beginning of the functions associated with this part instance.

Return Type

Physical Function Iterator

Syntax

```
IPhysPartInst_beginFunc <PhysPartInst>
```

Parameters

Parameter	Type	Description
`PhysPartInst`	`Physical Part Instance`	Specifies the object representing the physical part instance in the design. This parameter is required.

Examples

```
IPhysPartInst_beginFunc $PhysPartInst
```

Related Commands

getServer

IServer_findDesign

getPhysPartInst

IPhysPartInst_beginProp

This method is used to get an iterator representing the start of physical properties on a physical part instance.

Return Type

Property Iterator

Syntax

```
IPhysPartInst_beginProp <PhysPartInst>
```

Parameters

Parameter	**Type**	**Description**
`PhysPartInst`	`Physical Part Instance`	It specifies the object representing the physical part instance in the design This parameter is required.

Examples

```
IPhysPartInst_beginProp $PhysPartInst
```

Related Commands

getServer

IServer_findDesign

getPhysPartInst

IPhysPartInst_findProp

Returns a string containing the value of the specified property. If the property is not found, an empty string is returned.

Return Type

Property Value

Syntax

```
IPhysPartInst_findProp <PhysPartInst> <PropName> <Hierarchical>
```

Parameters

Parameter	Type	Description
`PhysPartInst`	`Physiacl Part Instance`	Specifies the object representing the physical part instance in the design. This parameter is required.
`PropName`	`String`	The string representing the property name to be found on this part instance. This parameter is required.
`Hierarchical`	`INT`	This is a Boolean parameter. If this is FALSE, the part definition is not searched for the property. If the value of this variable is true, part definition is also searched. This parameter is required.

Examples

```
IPhysPartInst_findProp $PhysPartInst $PropName $Hierarchical
```

Related Commands

getServer

IServer_findDesign

getPhysPartInst

IPhysPartInst_isSectionUsed

Checks whether the specified section number is used or not.

Return Type

INT

Syntax

```
IPhysPartInst_isSectionUsed <PhysPartInst> <iSection>
```

Parameters

Parameter	Type	Description
`PhysPartInst`	`Physical Part Instance`	Specifies the object representing the physical part instance in the design. This parameter is required.
`iSection`	`INT`	An integer representing the section number. This parameter is required.

Examples

```
IPhysPartInst_isSectionUsed $PhysPartInst $iSection
```

Related Commands

getServer

IServer_findDesign

getPhysPartInst

IPhysPartInst_name

Returns the name of the specified part instance.

Return Type

Reference Designator

Syntax

```
IPhysPartInst_name <PhysPartInst>
```

Parameters

Parameter	Type	Description
`PhysPartInst`	`Physical Part Instance`	Specifies the object representing the physical part instance in the design. This parameter is required.

Examples

```
IPhysPartInst_name $PhysPartInst
```

Related Commands

getServer

IServer_findDesign

getPhysPartInst

IPhysPartInst_usedSections

Returns an integer value representing the number of sections used in this part instance.

Return Type

INT

Syntax

```
IPhysPartInst_usedSections <PhysPartInst>
```

Parameters

Parameter	Type	Description
`PhysPartInst`	`Physical Part Instance`	Specifies the object representing the physical part instance in the design. This parameter is required.

Examples

```
IPhysPartInst_usedSections $PhysPartInst
```

Related Commands

getServer

IServer_findDesign

getPhysPartInst

IPhysPin_getLogicalPin

Returns the logical pin associated with the specified physical pin instance. The value returned is an instance terminal bit context for this physical pin instance.

Return Type

Instance Terminal Bit Context

Syntax

```
IPhysPin_getLogicalPin <PhysPin>
```

Parameters

Parameter	Type	Description
`PhysPin`	`Physical Pin Instance`	Specifies the object representing the physical pin instance in the design. This parameter is required.

Examples

```
IPhysPin_getLogicalPin $PhysPin
```

Related Commands

getServer

IServer_findDesign

getPhysPinInst

IPhysPin_getPinTypeStr

Returns the pin type of the specified physical pin instance.
The possible return values include:
IN, OUT, BI, POWER, GROUND, UNSPEC, NC

Return Type

Pin Type

Syntax

```
IPhysPin_getPinTypeStr <PhysPin>
```

Parameters

Parameter	Type	Description
`PhysPin`	`Physical Pin Instance`	Specifies the object representing the physical pin instance in the design. This parameter is required.

Examples

```
IPhysPin_getPinTypeStr $PhysPin
```

Related Commands

getServer

IServer_findDesign

getPhysPinInst

IPhysPin_getProperty

Returns the value of the specified property for the physical pin instance.

Return Type

Property Value

Syntax

```
IPhysPin_getProperty <PhysPin> <name>
```

Parameters

Parameter	Type	Description
`PhysPin`	`Physical Pin Instance`	Specifies the object representing the physical pin instance in the design. This parameter is required.
`name`	`String`	The String representing the property name. This parameter is required.

Examples

```
IPhysPin_getProperty $PhysPin $name
```

Related Commands

getServer

IServer_findDesign

getPhysPinInst

IPhysPin_isConnected

Checks if the physical pin is connected to a net or not. Returns True if the pin is connected, else returns False.

Return Type

INT

Syntax

```
IPhysPin_isConnected <PhysPin>
```

Parameters

Parameter	Type	Description
`PhysPin`	`Physical Pin Instance`	Specifies the object representing the physical pin instance in the design. This parameter is required.

Examples

```
IPhysPin_isConnected $PhysPin
```

Related Commands

getServer

IServer_findDesign

getPhysPinInst

IPhysPin_name

Returns the physical pin name of the specified physical pin instance.

Return Type

Physical Pin Name

Syntax

```
IPhysPin_name <PhysPin>
```

Parameters

Parameter	Type	Description
`PhysPin`	`Physical Pin Instance`	Specifies the object representing the physical pin instance in the design. This parameter is required.

Examples

```
IPhysPin_name $PhysPin
```

Related Commands

getServer

IServer_findDesign

getPhysPinInst

IPhysPin_physNet

Returns the physical net connected to specified physical pin instance.

Return Type

Physical Net

Syntax

```
IPhysPin_physNet <PhysPin>
```

Parameters

Parameter	Type	Description
`PhysPin`	`Physical Pin Instance`	Specifies the object representing the physical pin instance in the design. This parameter is required.

Examples

```
IPhysPin_physNet $PhysPin
```

Related Commands

getServer

IServer_findDesign

getPhysPinInst

IPhysPin_physPartInst

Returns the physical part instance that contains the specified physical pin instance.

Return Type

Physical Part Instance

Syntax

```
IPhysPin_physPartInst <PhysPin>
```

Parameters

Parameter	Type	Description
`PhysPin`	`Physical Pin Instance`	Specifies the object representing the physical pin instance in the design. This parameter is required.

Examples

```
IPhysPin_physPartInst $PhysPin
```

Related Commands

getServer

IServer_findDesign

getPhysPinInst

IPhysPin_spath

Returns the system canonical path of the physical pin instance in the design.

Return Type

Canonical Path

Syntax

```
IPhysPin_spath <PhysPin>
```

Parameters

Parameter	Type	Description
`PhysPin`	`Physical Pin Instance`	Specifies the object representing the physical pin instance in the design. This parameter is required.

Examples

```
IPhysPin_spath $PhysPin
```

Related Commands

getServer

IServer_findDesign

getPhysPinInst

more

Checks if there are elements that are yet to be iterated. Returns a boolean value of True or False. If there are more elements to be iterated, the return value is True, else False.

Return Type

Boolean

Syntax

```
more
```

Examples

```
# more command used on the physical pin iterator, $physPinIter,
# checks if any elements are left to be iterated over and returns 0 or 1.
# The returned value is saved to the variable val.
set val [$physPinIter more]
```

Related Commands

IPhysPartDefn_beginPartInst

IPhysNet_beginPin

IPhysFunc_beginPin

createPhysPartDefnIter

createIPhysNetIter

PhysPartInst_getPhysPartDefn

Returns the physical part definition object.

Return Type

Physical Part Definition

Syntax

```
PhysPartInst_getPhysPartDefn <PhysPartInst>
```

Parameters

Parameter	Type	Description
`PhysPartInst`	`Physical Part Instance`	Specifies the object representing the physical part instance in the design. This parameter is required.

Examples

```
PhysPartInst_getPhysPartDefn $PhysPartInst
```

Related Commands

getServer

IServer_findDesign

getPhysPartInst

7

Schematic Tools

addAlias

It adds an alias object to the page. Alias objects short two named nets.

Return Type

INT

Syntax

```
addAlias <lib> "<cell>" <view> ?-pg <spath_to_page>? ?-r <rotation_in_degrees>? ?-m
<mirror_type>? -pos [list x y] ?-tr [list <transform_matrix>]?
```

Parameters

Parameter	Type	Description
`lib`	`STRING`	Name of the library in which the component resides. <lib><cell><view> are the first three parameters of the addAlias command. This parameter is required.
`cell`	`STRING`	Name of the cell in the specified library. Enclose within double quotes (" "). This parameter is required.
`view`	`INT`	View name of the component to be added. Valid format if 'sym_<number>'. This parameter is required.
`spath_to_page`	`STRING`	sPath representing the page This parameter is optional. Default value is `current page`.
`rotation_in_degrees`	`INT`	Rotation in degrees This parameter is optional.
`[list <transform_matrix>]`	`LIST`	Transformation matrix to indicate mirroring and rotation This parameter is optional. Default value is `none`.
`[list x y]`	`LIST`	x and y coordinates of the alias on schematic page This parameter is required.
`mirror_type`	`INT`	Mirror rotation of the alias to be placed, whether horizontal or vertical. This parameter is optional.

Examples

```
#command to store the active page name to the variable called pageSPath
set pageSPath sch::dbGetActivePageSPath
#without optional parameters
addAlias -pos [list 3400 3050]
#with optional parameters
addAlias -pg $pageSPath -r 90 -m 0 -pos [list 3300 3850]
```

addBypass

Adds a bypass circuit on the schematic page.

Return Type

NONE

Syntax

```
addBypass -sym [list <lib> <cell> <view> <cellname in uppercase>] -net [list
<pwr_pin_name> <pwr_net_name> <gnd_pin_name> <gnd_net_name>] ?-n [<quantity>]? -parent
<value> -refdesstart <value> -caption {value} ?-pg [<page_name>]? -pos [list <x-
coordinate> <y-coordinate>] -key [list <properties>] -powerSym [list <lib> <cell>
<view>] -gndSym [list <lib> <cell> <view>]
```

Parameters

Parameter	Type	Description
`[list <lib> <cell> <view> <cellname in uppercase>]`	`STRING`	Capacitor library, cell, and the version number. This parameter is required.
`[list <pwr_pin_name> <pwr_net_name> <gnd_pin_name> <gnd_net_name>]`	`STRING`	Pin name and net name of the power and ground connections. This parameter is required.
`quantity`	`INT`	Number of capacitors in the bypass circuit. This parameter is optional. Default value is `1`.
`value`	`STRING`	Name of the parent design. This parameter is required.

`value`	`INT`	Reference designator of the starting capacitor. This parameter is required.
`{value}`	`STRING`	Caption for the bypass circuit. This parameter is required.
`page_name`	`STRING`	Name of the page to which the element is being added. This parameter is optional. Default value is `Current page is used as the default value.`.
`[list <x-coordinate> <y-coordinate>]`	`INT`	Top-left XY coordinates of the grid. This parameter is required.
`[list <properties>]`	`STRING`	Key properties of the capacitor. This parameter is required.
`[list <lib> <cell> <view>]`	`STRING`	The lib:cell:view value of the power symbol. This parameter is required.
`[list <lib> <cell> <view>]`	`STRING`	The lib:cell:view value of the ground symbol. This parameter is required.

Examples

```
addBypass -sym [list passives cap sym_1 CAP] -net [list B GND3 A GND_2] -n 1 -parent I2
-refdesstart 1 -caption {Bypass rails of $Parent} -pg @worklib.workshop1(tbl_1):page(2)
-pos [list 9091 8450] -key [list {PACK_TYPE=0603} {PART_NAME=CAP} {PART_NUMBER=CDN-CAP-
0001} {TOLERANCE=5%} {VALUE=330pF} {VOLTAGE=50V}] -powerSym [list standard gnd sym_1] -
gndSym [list standard gnd sym_1] -distanceValue 3 -distanceUnits Mils
```

Related Commands

dbGetActivePageSPath

addComponent

It adds a component on the schematic page. Returns "incomplete" or "addComponent command exited with an error" in case of failure. On success, does not return anything.
Library specified should be part of the project.

Return Type

NONE

Syntax

```
addComponent <lib> "<cell>" <view> "<cell name in uppercase>" ?-n 1? -key [list
<properties>] ?-r<value>? ?-m<value>? ?-pg<page_name>? -pos [list <value>] ?-o?
```

Parameters

Parameter	Type	Description
`lib`	`STRING`	Name of the library in which the component resides. <lib><cell><view> are the first three parameters of the addComponent command. This parameter is required.
`cell`	`STRING`	Name of the cell in the specified library. Enclose within double quotes (" "). This parameter is required.
`view`	`INT`	View name of the component to be added. Valid format if 'sym_<number>'. This parameter is required.
`partname`	`STRING`	Partname to be added in uppercase. Enclose within double quotes (" "). This parameter is required.

`-n`	`INT`	Ignored. Unused parameter. This parameter is optional. Default value is `1`.
`-key`	`STRING`	Key properties of the component and their corresponding values. Key properties are VALUE, PACK_TYPE, and PART_NUMBER. This parameter is required.
`-r`	`INT`	Rotates the offpage connector by the specified value. Valid rotation values are multiple of 90. Rotation is measured in degrees, positive value for clockwise direction and negative value for the anticlockwise direction. This parameter is optional.
`-m`	`INT`	Currently, this parameter takes only 1 value, '0'. This parameter is optional.
`-pg`	`INT`	Name of the page where the component is to be placed. This parameter is optional. Default value is `current page`.
`-pos`	`INT`	The X-Y coordinates of the left-most grid where the component is to be placed. This parameter is required.
`-o`	`BOOL`	This is to be specified in tcl mode for no popup. This parameter is optional.

Examples

```
addComponent passives "cap" sym_1 "CAP" -n 1 -key [list {VOLTAGE=50V=1}
{TOLERANCE=5%=1} {VALUE=330pF=1} {PACK_TYPE=0603=1} {PART_NUMBER=CDN-CAP-0001=1}
{PART_NAME=CAP=0}] -pg @worklib.workshop1(tbl_1):page(1) -pos [list 6650 7900]
#command to add component 'NC7SB3157' from the 74x library
#on page 1 of the schematic
addComponent 74x "nc7sb3157" sym_2 "NC7SB3157" -n 1 -key [list VALUE=NC7SB3157P6XG
PACK_TYPE=SC70 PART_NAME=NC7SB3157 ] -pg page(1) -r 0 -m 0 -pos [list 4350 3850]
#command to add '7410' part at specified position
#(Without optional parameters)
addComponent combinatorial "7410" sym_1 "74AC10" -key [list {PART_NAME=74AC10=1}
{TRADE_CODE=74AC10=1} {PART_NUMBER=RYT3180010/C=1} ] -pos [list 12250 7850]
```

Related Commands

addPort

addAlias

addPower

addOffPage

addLib

dbGetActivePageSPath

addConnector

Draws a block arrow to connect block shapes and drawing objects.
Returns 0 on success and "incomplete" or "command exited with an error" on failure.

Examples for <shape_name> :- Wire, Wire left arrow, Wire right arrow, Wire double arrow, Bus, Bus left arrow, Bus right arrow, Bus double arrow, Bundle, Bundle right arrow, Bundle left arrow, Bundle double arrow, Arrow pointed, Arrow diamond, PCIe double arrow.

Return Type

INT

Syntax

```
addConnector <shape name> ?-pg <page_name>? -b [list <x-coordinate> <y-coordinate>]
[list <x-coordinate> <y-coordinate>]
```

Parameters

Parameter	Type	Description
`shape name`	`STRING`	The name of the arrow shape to be added. This parameter is required.
`-pg <page_name>`	`STRING`	Name of the page where the connector is to be added. This parameter is optional. Default value is `current page`.
`-b [list <x-coordinate> <y-coordinate> <x-coordinate> <y-coordinate>]`	`INT`	The leftmost X-Y coordinates and the rightmost X-Y coordinates of the arrow shape This parameter is required.

Examples

```
#Command to add block arrow with optional parameters
addConnector LeftWireArrow -pg page(1) -b [list 3970 10110] [list 13870 10110]

#Command to add block arrow without optional parameters
addConnector LeftWireArrow -b [list 3970 10110] [list 13870 10110]
```

Related Commands

dbGetActivePageSPath

addCustomVar

It adds a custom variable to the variable list. Tool Defined variables will be added to every project created while Project Defined variables are specific to the project.

Return Type

NONE

Syntax

```
addCustomVar -name <variable_name> -value <variable_value> -type <value>
```

Parameters

Parameter	Type	Description
`-name`	`STRING`	Variable name. Variable name must be specified in uppercase. This parameter is required.
`-value`	`STRING`	Variable value. Values must be enclosed within double quotes (" "). This parameter is required.
`-type`	`STRING`	Supported values are "Tool Defined" or "Project Defined". Value must be enclosed within double quotes (" "). This parameter is required.

Examples

```
addCustomVar -name LKJ -value "ewq" -type "Project Defined"
addCustomVar -name CON_ROOT_LIB -value "worklib" -type "Project Defined"
```

addDirectiveValue

It adds a directive within a given section in the project. A project should be opened for this command to work. It returns 1 on success and 0 on failure.

Return Type

INTEGER

Syntax

```
cps::addDirectiveValue <sectionName> <directiveName> <directiveValue> <valueType>
```

Parameters

Parameter	Type	Description
`sectionName`	`STRING`	Section within the project where the new directive is added This parameter is required.
`directiveName`	`STRING`	Name of the directive This parameter is required.
`directiveValue`	`STRING`	Value of the directive This parameter is required.
`valueType`	`STRING`	Type of the directive value. Allowed types are BOOL, STRING, INT, DOUBLE, LONG This parameter is required.

Examples

```
cps::addDirectiveValue CANVAS DWG_ID "TEST-001" STRING.
```

Related Commands

getDirectiveValue

getSections

getDirectives

setDirectiveValue

addDock

Adds a docked panel widget at the location specified.
The dockWidgetHandle is returned when a dock widget is created using cpCommon::dockedHybridInit Tcl command.
The accepted values for dockArea are- ::cps::DA_LEFT, :cps::DA_RIGHT, and :cps::DA_BOTTOM.

Return Type

INT

Syntax

```
cps::addDock <dockWidgetHandle> <dockArea> <widgetOrientation> ?<visibility>?
```

Parameters

Parameter	Type	Description
`dockWidgetHandle`	`INT`	Dock widget handle is the return value of the dock widget created. This parameter is required.
`dockArea`	`INT`	Where to place the docked panel. Valid values are ::cps::DA_LEFT,:cps::DA_RIGHT, and :cps::DA_BOTTOM This parameter is required.
`widgetOrientation`	`STRING`	Orientation of the docked panel. Not used currently. It should always be ::cps::WO_VERT. This parameter is required.
`visibility`	`INT`	Visibility state of the docked widget, if 0, the docked widget is not displayed. If 1, then the widget is displayed. This parameter is optional. Default value is `1`.

Examples

```
# use cpCommon::dockedHybridInit command to generate the dockWidgetHandle
set dockHandle [cpCommon::dockedHybridInit {CADENCE} {testtab02} {www.cadence.com}]
puts $dockHandle
# This will return following output as dock widget handle, which will be passed to
addDock Tcl command
# 87

# Add docked panel to Left area of application.
cps::addDock $dockHandle $::cps::DA_LEFT $::cps::WO_VERT 1
```

Related Commands

dockedHybridInit

addEllipse

Draws an ellipse shape on the schematic page.

Return Type

INT

Syntax

```
addEllipse ?-pg <page_name>? [list <topLeft x-coordinate> <topLeft y-coordinate] [list
<bottomRight x-coordinate> <bottomRight y-coordinate>]
```

Parameters

Parameter	Type	Description
`-pg <page_name>`	`STRING`	Name of the page where the ellipse is to be added. This parameter is optional. Default value is `current page`.
`[list <topLeft x-coordinate> <topLeft y-coordinate>] [list <bottomRight x-coordinate> <bottomRight y-coordinate>]`	`INT`	Top-left XY-coordinates and bottom-right XY-coordinates of the rectangle in which ellipse is drawn. This parameter is required.

Examples

```
#Command to add ellipse with optional parameter
addEllipse -pg page(1) 7220 3590 11470 7220

#Command to add ellipse without optional parameter
addEllipse 7220 3590 11470 7220
```

Related Commands

dbGetActivePageSPath

addImage

Adds an image on the schematic page.
List of supported image formats: bmp, cur, dds, gif, icns, ico, jpeg, jpg, pbm, pgm, png, ppm, svg, svgz, tga, tif, tiff, wbmb, webp, xbm, xpm

Return Type

INT

Syntax

```
addImage ?[-pg <page_name>]? <topleft_x-coordinate> <topleft_y-coordinate> -img
<image_pathname> -override <choice>
```

Parameters

Parameter	Type	Description
`page_name`	`STRING`	Name of the page on which the image is to be placed. This parameter is optional. Default value is `Current page`.
`topleft_x-coordinate`	`INT`	The Top-left X-coordinates of the grid. This parameter is required.
`topleft_y-coordinate`	`INT`	The Top-left Y-coordinates of the grid This parameter is required.
`image_pathname`	`STRING`	Image pathname to be specified within curly braces { }. This parameter is required.

Examples

```
addImage -pg page(1) 4000 3000 -img {C:/images/cad_logo2.gif} -override true

#"-pg page(1)" signifies the name of the page where image is be added,
#"4000 3000" signifies the top-left x and y coordinates of the image on the
#schematic page, "-img { }" signifies the local path where image is stored,
#"-override" says older image will be removed if YES, or both older will not
#be removed if NO.
```

Related Commands

dbGetActivePageSPath

addLib

It adds a library to the project libraries. It returns an empty string on success, else an error message string on failure.

Return Type

STRING

Syntax

```
addLib <dir_path> <lib_name>
```

Parameters

Parameter	Type	Description
`dir_path`	`STRING`	Path to the directory. Enclose the path within double quotes "". This parameter is required.
`lib_name`	`STRING`	Name of the library. This parameter is required.

Examples

```
#command to add library 'doc' in the project.
addLib "D:/designs/archive_libs/doc" doc
```

addMenuToMenuBar

Adds a menu to the application menu bar. The new menu gets added to the end unless "beforeMenu" is specified. Adding the same menu more than once is not allowed. Returns 1 on the successful operation, otherwise, returns 0. On failure, an error message is returned.

Return Type

INT

Syntax

```
addMenuToMenuBar <menuName> <iconPath> <enabled> ?beforeMenu? ?context?
```

Parameters

Parameter	Type	Description
`menuName`	`STRING`	Name of the new menu item to be added to the menu bar This parameter is required.
`iconPath`	`STRING`	This parameter is reserved for future use and should be passed the empty list - {} This parameter is required.
`enabled`	`BOOLEAN`	Default state of the menu item. 1 enables the menu, and 0 disables the menu This parameter is required.
`beforeMenu`	`STRING`	Name of the menu in the menubar before which the menu entry will be added. If this is not provided, the new menu is added as the last entry in the menubar This parameter is optional.
`context`	`STRING`	The menu item will be added to the provided context's menu bar. This parameter is optional. Default value is `sch`.

Examples

```
# This command adds the menu "Custom Tools" before
# the "Help" menu in sch context menu bar
addMenuToMenuBar "Custom Tools" {} 1 Help sch
```

Related Commands

addActionToMenu

addActionToMenuId

addActionToMenuIdEx

addMenuToMenuId

addMenuToMenuName

addSeparatorToMenu

deleteSeparatorFromMenu

getMenuId

getResourceFullPath

addMenuToMenuBarEx

Adds a menu to the application menu bar. The new menu gets added to the end unless "beforeMenu" is specified. This command is an extension of addMenuToMenuBar and it allows to have same named multiple menus in either same or different contexts in menubar, here the displayName assigned can be same for multiple menus only menuName has to be different. Returns 1 on the successful operation, otherwise, returns 0. On failure, an error message is returned.

Return Type

INT

Syntax

```
addMenuToMenuBarEx <menuName> <displayName> <iconPath> <enabled> ?beforeMenu? ?context?
```

Parameters

Parameter	Type	Description
`menuName`	`STRING`	Menu name of the new menu item to be added to the menu bar. This argument is unique for every menu. This parameter is required.
`displayName`	`STRING`	Display name of the new menu item to be added to the menu bar. This argument can be same for multiple menus.. This parameter is required.
`iconPath`	`STRING`	This parameter is reserved for future use and should be passed the empty list - {} This parameter is required.
`enabled`	`BOOLEAN`	Default state of the menu item. 1 enables the menu, and 0 disables the menu This parameter is required.
`beforeMenu`	`STRING`	Name of the menu in the menubar before which the menu entry will be added. If this is not provided, the new menu is added as the last entry in the menubar This parameter is required.
`context`	`STRING`	The menu item will be added to the provided context's menu bar. This parameter is optional. Default value is `sch`.

Examples

```
# Following commands adds custom menus named "custom menu 1" in menubar before #"Help"
menu in both global and schematic contexts
addMenuToMenuBarEx "custom menu1" "custom menu 1" {} 1 "Help" "cps"
addMenuToMenuBarEx "custom menu2" "custom menu 1" {} 1 "Help" "sch"
```

Related Commands

addMenuToMenuBar

addActionToMenu

addActionToMenuEx

addActionToMenuEx2

addActionToMenuId

addActionToMenuIdEx

addMenuToMenuId

addMenuToMenuName

addMenuToMenuNameEx

addSeparatorToMenu

deleteSeparatorFromMenu

getMenuId

getResourceFullPath

addMenuToMenuNameEx

It adds a pop-up menu to an existing menu. This command helps in adding same named pop-up menus multiple times either in same or different contexts. It returns the ID of the pop-up menu on success and 0 on failure.

Return Type

INT

Syntax

```
addMenuToMenuNameEx <parentMenuName> <menuName> <displayName> <iconPath> <enabled>
```

Parameters

Parameter	Type	Description
`parentMenuName`	`STRING`	Name of the parent menu in which the sub-menu is added. Menu IDs of an existing menu can be obtained using the getMenuId command. This parameter is required.
`menuName`	`STRING`	The name of the pop-up menu to be added. This has to be unique for each pop-up menus. This parameter is required.
`displayName`	`STRING`	The display name of the pop-up menu to be added. This can be same for multiple pop-up menus. This parameter is required.
`iconPath`	`STRING`	This parameter is reserved for future use and must be an empty list {}. This parameter is required.
`enabled`	`BOOLEAN`	Default state of the menu item. 1 enables the menu, and 0 disables the menu. This parameter is required.

Examples

```
# These commands will add "pop-up-menu" named pop-up menus in both Help and # Tools
menu in menubar.
addMenuToMenuNameEx "Help" "pop-up1" "pop-up-menu" {} 1
addMenuToMenuNameEx "Tools" "pop-up2" "pop-up-menu" {} 1
```

Related Commands

addMenuToMenuName

addActionToMenu

addActionToMenuEx

addActionToMenuEx2

addActionToMenuId

addActionToMenuIdEx

addMenuToMenuId

addMenuToMenuName

addSeparatorToMenu

deleteSeparatorFromMenu

getMenuId

getResourceFullPath

addNote

Adds a text note at the specified location on the schematic page.

Return Type

INT

Syntax

```
addNote ?-pg <page_name>? -pos <position> -s <new_text>
```

Parameters

Parameter	Type	Description
`page_name`	`STRING`	The page where the note is to be added This parameter is optional. Default value is `current page`.
`position`	`LIST`	The coordinate of the top-left corner of the note item. Coordinates are represented as a list of two integers representing X, Y. This parameter is required.
`new_text`	`STRING`	The text to be added. This parameter is required.

Examples

```
addNote -pos [list 6553 5696] -s {Another text added}
addNote -pg page(1) -pos [list 6553 5696] -s {This is a new Note}
```

Related Commands

dbGetActivePageSpath

addOffPage

It adds an instance of offpage connector of the specified type in the specified location. Only the offpage connectors defined and available in the special body pallet are instantiated on the canvas.

Return Type

INT

Syntax

```
addOffPage <type> | ?-lib <libraryName> -cell <cellName> -view <viewName>? ?-pg
<pageName>? ?-r <rotationValue>? ?-m <mirrorType>? -pos <position>
```

Parameters

Parameter	Type	Description
`type`	`STRING`	Type of offpage connector. Valid values are input, output, and inout. This parameter is optional.
`pageName`	`STRING`	Name of the page on which the offpage connector is to be added. If this parameter is not specified, root design is used as the design name. This parameter is optional. Default value is `Current Page`.
`rotationValue`	`INT`	Rotates the offpage connector by the specified value. Valid rotation values are multiple of 90. Rotation is measured in degrees, positive value for clockwise direction and negative value for the anticlockwise direction. This parameter is optional.
`mirrorType`	`INT`	Currently, this parameter takes only 1 value, '0'. This parameter is optional.
`position`	`LIST`	X-Y coordinates where the offpage connector is to be placed on the canvas. This parameter is required.

Examples

```
#command adds input offpage component at specified location.
addOffPage -input -pos [list 5950 6050]
#command adds input offpage, component at specified location,
#at the specified page with the specified rotation value.
addOffPage -input -pg @worklib.workshop1(tbl_1):page(2) -r 0 -m 0 -pos [list 5950 6050]
#command adds I/O offpage, component at specified location,
#at the specified page with the specified rotation value.
addOffPage -io -pg @worklib.tr1(tbl_1):page(1) -r 0 -m 0 -pos [list 2800 4250]
```

Related Commands

addPort

addAlias

addPower

addComponent

dbGetActivePageSPath

preferenceSpecialBodies

addPageFromPageToolBar

Adds a page below to the current page

Return Type

BOOL

Syntax

```
sch::addPageFromPageToolBar
```

Examples

```
Tcl&gt; sch::addPageFromPageToolBar
createItem {@worklib.workshop1(tbl_1):name(2)} SCH PAGE
0
Tcl&gt;
```

addPort

It adds an instance of the port connector of the specified type at the specified location. Only the port connectors defined and available in the special symbols palette are instantiated on the canvas. It returns 0 on success and "incomplete" or "command exited with an error" on failure.

Return Type

INT

Syntax

```
addPort <type> | ?-lib <libraryName> -cell <cellName> -view <viewName>? ?-pg
<pageName>? ?-r <rotationValue>? ?-m <mirrorType>? -pos [list <x-coordinate> <y-
coordinate>]
```

Parameters

Parameter	Type	Description
`type`	`STRING`	Type of port. Types are either input, output, or io This parameter is optional.
`pageName`	`STRING`	Name of the page on which the port connector is to be added. If this parameter is not specified, root design is used as the design name. This parameter is optional. Default value is `Current Page`.
`rotationValue`	`INT`	Rotates the offpage connector by the specified value. Valid rotation values are multiple of 90. Rotation is measured in degrees, positive value for clockwise direction and negative value for the anticlockwise direction. This parameter is optional.
`mirrorType`	`INT`	Currently, this parameter takes only '0' in the value. This parameter is optional.
`position`	`LIST`	X-Y coordinates where the offpage connector is to be placed on the canvas. This parameter is required.

Examples

```
#command adds input port component at specified location.
addPort -input -pos [list 5950 6050]
#command adds output port, component at specified location,
#at the specified page with the specified rotation value.
addPort -output -pg @worklib.workshop1(tbl_1):page(2) -r 0 -m 0 -pos [list 5950 6050]
#command adds io port, component at specified location,
#at the specified page with the specified rotation value.
addPort -io -pg @worklib.tr1(tbl_1):page(1) -r 0 -m 0 -pos [list 2800 4250]
#command adds io port, component with lib cell view
addPort "standard" "outport" "sym_1" -pg @worklib.top(tbl_1):page(1) -r 0 -m 0 -pos
[list 8050 9750]
```

Related Commands

addOffPage

addAlias

addPower

addComponent

dbGetActivePageSPath

addPower

It adds ground and power symbols on the schematic page. It returns NULL on success and "incomplete" or "command exited with an error" on failure.

Return Type

NONE

Syntax

```
addPower <lib> "<cell>" <view> -i [list <key properties>] ?-g? ?-pg <page_name>? ?-r
<rotational_value>? ?-m <mirror_value>? -pos [list <x-coordinate> <y-coordinate>]
```

Parameters

Parameter	Type	Description
`lib`	`STRING`	Name of the library in which the component resides. This parameter is required.
`cell`	`STRING`	Name of the required cell in the specified library. Enclose within double quotes (""). This parameter is required.
`view`	`STRING`	View name of the component to be added. Valid format if 'sym_<number>'. This parameter is required.
`key_properties`	`STRING`	Key properties of the component to be added. Key properties include the signame and the voltage properties. This parameter is required.
`-g`	`STRING`	Indicates that the symbol added to the schematic page is a ground symbol. This parameter is optional.

`-pg`	`STRING`	Name of the page on which the off-page connector is to be added. If this parameter is not specified, root design is used as the design name.he page where the component is to be added. This parameter is optional. Default value is `current page`.
`-r`	`INT`	Rotates the offpage connector by the specified value. Valid rotation values are multiple of 90. Rotation is measured in degrees, positive value for clockwise direction and negative value for the anticlockwise direction. This parameter is optional.
`-m`	`INT`	Mirror rotation of the component to be placed, whether horizontal or vertical. Currently, valid value is, '0'. This parameter is optional.
`-pos`	`INT`	The X-Y coordinates of the rightmost grid where the component is to be placed. This parameter is required.

Examples

```
#adds pvcc part at the specified location.
addPower standard "pvcc" sym_1 -i [list signame=PVCC voltage=5] -pg
@worklib.workshop1(tbl_1):page(2) -r 0 -m 0 -pos [list 7242 8158]
#adds GND pat at the specified location.
addPower standard "gnd" sym_1 -i [list signame=GND voltage=0] -g -pos [list 7242 8158]
```

Related Commands

addOffPage

addAlias

addComponent

addPort

dbGetActivePageSPath

addPreferredPart

Adds a preferred component to an existing component in the design.

Return Type

NONE

Syntax

```
addPreferredPart <lib> "<cell>" <view> "<cell name in uppercase>" -n "1" -key [list
<key_properties>] -i [list <injected_properties>]
```

Parameters

Parameter	Type	Description
`lib`	`STRING`	Name of the library in which the component resides. This parameter is required.
`cell`	`STRING`	Name of the cell in the specified library. Enclose within double quotes (" "). This parameter is required.
`view`	`STRING`	Version number of the component to be added. This parameter is required.
`cell_name_in_uppercase`	`STRING`	Name of the cell to be added in uppercase. Enclose within double quotes (" "). This parameter is required.
`-n "1" -key [list <key_properties>]`	`STRING`	List of key properties. This parameter is required.
`list <injected_properties`	`STRING`	List of injected properties. This parameter is required.

Examples

```
#Command to create a preferred capacitor which resides in library "discrete"
#inside cell name "cap" and its version is "sym_2" with key properties mentioned
#as "-key[list]" and with injected properties as mentioned "-i [list]"

addPreferredPart discrete "cap" sym_2 "CAP" -n "1" -key [list "VOLTAGE=50V"
"MATERIAL=X7R-CERM" "TOLERANCE=10%" "VALUE=0.0015UF" "PACK_TYPE=0402-1" "PART_NAME=CAP"
] -i [list "VOLTAGE=50V" "QUAD_MODEL=CAP_0.0015UF" "TOLERANCE=10%" "JEDEC_TYPE=CAP-
0402-HP55" "VALUE=0.0015UF" "ALT_SYMBOLS=(CAP-0402-HP55-FLEX-CL-P25-3,CAP-0402-HP55-
FLEX-LPI-P05-4)" "PART_NUMBER=132S0335" ]
```

Related Commands

createVariant

addProp

It adds a property with the value and display options to one or more selected objects.

Return Type

NONE

Syntax

```
addProp -name {value} -value {value} ?-type -<value>? -display <value>
```

Parameters

Parameter	Type	Description
`-name`	`STRING`	Name of the property. This parameter is required.
`-value`	`STRING`	Value of the property. This parameter is required.
`-type`	`INT`	Value type of property, for internal purpose. Valid values are 0,1,2,3 respectively for STRING, INT, DOUBLE and BOOL. This parameter is optional.
`-display`	`STRING`	Specify the display options. Valid values are valOnly, nameOnly, both, nodisp. For displaying only property value, only name, name and value both and nothing respectively. This parameter is required.

Examples

```
#command adds property with Boolean value type and no display
addProp -name {NO_DIFF_PAIR} -value 1 -type 3 -display nodisp
#command adds property with Integer value type and no display
addProp -name {TESTPROP} -value {testval} -type 1 -display nodisp
#command adds property with String value type, displaying value only
addProp -name {VOLTAGE} -value {5 V} -type 0 -display valOnly
```

Related Commands

modifyProp

deleteProp

selectObject

addRectangle

It draws a rectangle on the schematic page. The rectangle size is specified by the X-Y coordinates of the diagonal.

Return Type

INT

Syntax

```
addRectangle ?-pg <page_name>? <topLeft x-coordinate> <topLeft y-coordinate>
<bottomRight x-coordinate> <bottomRight y-coordinate>
```

Parameters

Parameter	Type	Description
`page_name`	`STRING`	Name of the page to which the rectangle is to be added. This parameter is optional. Default value is `current page`.
`topLeft x-coordinate topLeft y-coordinate`	`INT`	Top-left X-Y coordinates. This parameter is required.
`bottomRight x-coordinate bottomRight y-coordinate`	`INT`	Bottom-right X-Y coordinates. This parameter is required.

Examples

```
#command adds rectangle on the specified page at the
#specified top-left and bottom-right position
addRectangle -pg @worklib.workshop1(tbl_1):page(2) 13810 6920 14810 7920
#command adds rectangle on the selected page at the
#specified top-left and bottom-right position
addRectangle 13810 6920 14810 7920
```

Related Commands

dbGetActivePageSPath

addLine

addEllipse

addSeparatorToToolBar

It adds a separator to a toolbar section. There are three sections in the application toolbar:
0 - the left section
1 - the center section
2 - the right section
The schematic editor context is "sch".
It returns 1 on success and 0 on failure.

Return Type

BOOLEAN

Syntax

```
addSeparatorToToolBar <context> <section>
```

Parameters

Parameter	Type	Description
`context`	`STRING`	Editor context of the toolbar to which the separator is added This parameter is required.
`section`	`INT`	Specify the section number to which the separator is to be added. The supported section values are : '0' for Left section, '1' for Center section, and '2' for the Right section. This parameter is required.

Examples

```
#command to add a separator at the end of the center section
#of the toolbar.
addSeparatorToToolBar "sch" 1
```

Related Commands

addSeparatorToMenu

addSeparatorToContextMenu

addTable

Use addTable to place a table on the schematic page with given number of rows and columns on the specified page.

Return Type

NONE

Syntax

```
addTable ?-pg <page_spath>? ?-row <num_of_rows>? ?-column <num_of_column>? -pos
<pos_value>
```

Parameters

Parameter	**Type**	**Description**
`page_spath`	`STRING`	Specify spath of a schematic page. This parameter is optional. Default value is `Current Page`.
`num_of_rows`	`INT`	Number of rows required in the table. This parameter is optional. Default value is `4`.
`num_of_columns`	`INT`	Number of columns required in the table. This parameter is optional. Default value is `4`.
`pos_value`	`LIST`	X-Y coordinates of the top-left corner of the table. This parameter is required.

Examples

```
#command to add a 4x6 table on page 5 of design top
#on specified position.
addTable -pg @worklib.top(tbl_1):page(5) -row 4 -column 6 -pos [list 7700 6950]
#command to add a 5x6 table on current page of a
#design on specified position.
addTable -row 5 -column 6 -pos [list 2350 8200]
#command to add a 4x4 table on current page of a
#design on specified position
#default value of row and column is 4x4.
addTable -pos [list 2350 8200]
#pos [list 2350 8200] signifies the x-y coordinates of
#the top-left corner of the table
```

Related Commands

dbGetActivePageSPath

addToolItemToToolBar

It adds an action to the toolbar in the specified section. The top toolbar has 3 sections:
0 - Left section
1 - Center section
2 - Right section
It returns 1 on success, otherwise 0.

Return Type

BOOLEAN

Syntax

```
addToolItemToToolBar <toolItemName> <toolBarSection> <action> <context>
```

Parameters

Parameter	Type	Description
`toolItemName`	`STRING`	Unique string to identify a tool item. This parameter is required.
`toolBarSection`	`INT`	The section of the toolbar to which the tool item is to be added. This parameter is required.
`action`	`STRING`	Name of the action. This string needs to be enclosed within double quotes. This parameter is required.
`context`	`STRING`	The context in which the toolbar and the action is available. For schematic canvas actions, this should be "sch". This parameter is required.

Examples

```
#command to add the "Undo" action to the left section
#of the schematic editor toolbar.
addToolItemToToolBar undoToolButton 0 "Undo" "sch"
```

Related Commands

addSeparatorToToolBar

addToolItemToToolBarEx

addToolItemToToolBarEx

Adds an action to the toolbar in the specified section. The top toolbar has three sections:
0 - Left section
1 - Center section
2 - Right section
Command returns the action ID of new tool item on success, otherwise error is thrown.
The range of position argument starts from 0 which means first position from left most side of toolbar, if value of position is more than the number of tool items in toolbar section then tool item will be added at the end of the respective section.

Return Type

INT

Syntax

```
addToolItemToToolBarEx <toolItemName> <toolBarSection> <action> <context> ?position?
```

Parameters

Parameter	Type	Description
`toolItemName`	`STRING`	Unique string to identify a tool item This parameter is required.
`toolBarSection`	`INT`	The section of the toolbar to which the tool item is to be added. This parameter is required.
`action`	`STRING`	Name of the action. This string needs to be enclosed within double quotes. This parameter is required.
`context`	`STRING`	The context in which the toolbar and the action will be available. For schematic canvas actions, this should be "sch". This parameter is required.
`position`	`INT`	Specifies at which position from the left side of toolbar, the new item will be added. Value 0 of Position means left most position. If not specified, the new item gets added at the end of the toolbar. This parameter is optional. Default value is `None`.

Examples

```
#command to add the "Undo" action in the left section
#of the schematic editor toolbar at 1st position from left
addToolItemToToolBarEx undoToolButton 0 "Undo" 0
```

Related Commands

addToolItemToToolBar

addSeparatorToToolBar

archive

Archives the currently open project. By default, the archive file is created in the current project folder with the name <project name>_archive.zip

If the -compresscmd argument is provided, the current project is archived with the specifed name at the specified location. The path passed with -compresscmd must be the absolute path.

Return Type

NONE

Syntax

```
archive ? -compresscmd "<tar/zip command with absolute path to compressed file>"?
```

Parameters

Parameter	Type	Description
`-compresscmd`	`STRING`	-compresscmd argument is used to create a user-defined compressed tar or zip file. Specify the complete path and file name to be created in double-quotes. This parameter is optional.

Examples

```
# To archive the currently open project
archive

# To archive the currently open project to a specific location and name it compress.tar
archive -compresscmd "tar -cvf /home/user_xyz/compress.tar"

# To archive the currently open project to specific location and name it compress.zip
archive -compresscmd "cdszip -r C:/syscap/test/compress.zip"
```

ascend

It ascends the drawing hierarchy to the desired level. It returns 0 on success, else it returns an error message.

Return Type

STRING

Syntax

```
ascend ?level?
```

Parameters

Parameter	Type	Description
`level`	`INT`	The level to go up in the hierarchy. This parameter is optional. Default value is `1`.

Examples

```
#command to go 3 level up in the hierarchy
ascend 3
#command to go 1 level up in the hierarchy
ascend
```

Related Commands

descend

assignPinNumber

Associates pin numbers with pins. The new pin number must be compatible with the selected pin's section. If the selected object is not a pin, the command returns an error. This command works only for homogeneous multi-section parts and not on a single section part. The opt option resets any changes made in the assignment in an instance.

Return Type

STRING

Syntax

```
assignPinNumber <pinNumber> | <opt>
```

Parameters

Parameter	Type	Description
`pinNumber`	`STRING`	Pin number to be applied on the selected pin This parameter is required.
`opt`	`STRING`	Checks for the first available section in the same RefDes and assigns it to the selected instance This parameter is required.

Examples

```
assignPinNumber B5
```

assignPowerPins

This command is used to assign power to implicit power pins of a part. Different instances of the same part can be assigned different power on its pins.
To assign a power on an implicit pin, the net should have the VOLTAGE property assigned to it, that is it must be a power net.

Return Type

NONE

Syntax

```
assignPowerPins -refdes1 <refdesvalue> [list <powerpinname1> <newpowername1>] [list
<powerpinname2> <newpowername2>].... -refdes2 <refdesvalue> [list <powerpinname3>
<newpowername3>].....
```

Parameters

Parameter	Type	Description
`<refdesvalue>`	`STRING`	Refdes on whose implicit pins the power needs to be assigned This parameter is required.
`<powerpinname>`	`STRING`	Implicit pin of the part on which the power needs to be assigned This parameter is required.
`<newpowername>`	`STRING`	Power net which will be used to assign power on the implicit pin This parameter is required.

Examples

```
1. For example, a part ls00 with refdes U1 has power pins VCC and GND. To assign power
to VCC pin using a power net NETONE, give the following command -
assignPowerPins -refdes U1 [list VCC NETONE]

2. To assign NETONE to both VCC and GND pins -
assignPowerPins -refdes U1 [list VCC NETONE] [list GND NETONE]

3. In addition to this, assign power using power net NETTWO to pins of another instance
of ls00 with refdes U2 -
assignPowerPins -refdes U1 [list VCC NETONE] [list GND NETONE] -refdes U2 [list VCC
NETTWO] [list GND NETTWO]
```

assignShortcut

It assigns a key or the combination of keys as the shortcut keys to a command.

Return Type

NONE

Syntax

```
assignShortcut <command_name> <keyboard_shortcut>
```

Parameters

Parameter	Type	Description
`command_name`	`STRING`	Command name in straight double quotes (" "). This parameter is required.
`keyboard_shortcut`	`STRING`	The key combination to be assigned as a shortcut to the specified command. This value must be enclosed within straight double quotes (" "). This parameter is required.

Examples

```
#assigns "Ctrl+Shift+!" as shortcut keys for Add Note command
assignShortcut "Add &Note" "Ctrl+Shift+!"
```

auditDesign

Audits all the ECSet references in the design.

Return Type

NONE

Syntax

```
auditDesign
```

Examples

```
auditDesign
```

autoupdateparts

Updates the parts in the design from the reference library. This process updates only those parts, which does not require manual intervention.

Return Type

INT

Syntax

```
autoupdateparts
```

Examples

```
autoupdateparts
```

busEntry

It draws the bus taps on the selected bus trunk. If all parameters are not provided, the "Draw Multiple Bits" dialog is presented.

Return Type

INT

Syntax

```
busEntry <netBit {x1-pos y1-pos x2-pos y2-pos}> ...
```

Parameters

Parameter	Type	Description
`netBit`	`STRING`	Net bit to be tapped out This parameter is required.
`segment`	`LIST`	Wire segment position This parameter is required.

Examples

```
busEntry {ABCD&lt;0&gt;} [list 2650 3800 3350 3800] {ABCD&lt;1&gt;} [list 2650 3900
3350 3900]
busEntry {BUS_1&lt;1..0&gt;} [list 4000 10100 4000 10550] {BUS_1&lt;0&gt;} [list 4000
10250 4800 10250] {BUS_1&lt;1&gt;} [list 4000 10400 4800 10400]
#this will tap out bits 0 and 1 of the bus BUS_1&lt;1..0&gt; as BUS_1&lt;0&gt;
#and BUS_1&lt;1&gt; by drawing two horizontal wires.
#the wires are tapped out horizontally as the y-positions of
#the first and the second points are same for each of the wire.
```

callSkillFunc

Used for executing Constraint Manager SKILL procedures. Procedures with no arguments are not supported. For each parameter being passed, specify the parameter type. The type character for common types are:
string - x
int - i
double - d

Return Type

INT

Syntax

```
callSkillFunc <procedure_name> <specs> ?params?
```

Parameters

Parameter	Type	Description
`procedure_name`	`STRING`	The name of the Constraint Manager SKILL command or procedure to call. This parameter is required.
`specs`	`STRING`	Details of the number and type of parameters being passed to the SKILL function. For each parameter, parameter type is specified using a single type literal This parameter is required.
`params`	`LIST`	List of parameters to be passed to the Constraint Manager SKILL function. This parameter is optional.

Examples

```
#command to call a SKILL procedure "acm_afterCMLaunch"
#in System Capture with 2 string parameters.
set reportFileName {d:\Skill_data\report.txt}
set configFileName {d:\Skill_data\config.txt}
callSkillFunc "acm_afterCMLaunch" "xx" [list "$reportFileName" "$configFileName"]
```

Related Commands

loadSkillFile

callUnifiedSearch

This Tcl command is used to launch the Unified Search in System Capture. If a page is docked to a new workspace, and Add Component is launched from the View menu, the focus moves to Unified Search in the System Capture window.

Return Type

None

Syntax

```
cpSchT::callUnifiedSearch
```

Examples

```
cpSchT::callUnifiedSearch
```

captureScreenShot

Takes the screenshot of the tool window.

Return Type

NONE

Syntax

```
captureScreenShot
```

Examples

```
captureScreenShot
```

cdsCPSysRefreshKeywords

Forces a loading of all newly created TCL procedures in the session. This is useful when a new TCL procedure is created and it does not immediately become available in the currently open session.

Return Type

NONE

Syntax

```
cdsCPSysRefreshKeywords
```

Examples

```
# This example creates the following TCL procedure, and in same session when this
procedure is called, its suggestion won't be shown
proc newProc { } {
puts "New Procedure created"
}

# After below TCL command is executed, above procedure will be visible in the Command
Window's suggestion list.
cdsCPSysRefreshKeywords
```

changeBitNumber

It changes the bit number tapped out from a bus. The changed bit number dialog is shown if all parameters are not supplied.

Return Type

INT

Syntax

```
changeBitNumber [-pg <pageSPath>] [-taps [list <x-pos of bus-tap> <y-pos of bus tap>
<{current tap number}> <{new tap number}>] ... ]
```

Parameters

Parameter	Type	Description
`pageSPath`	`STRING`	SPath of a page This parameter is required.
`tap list`	`LIST`	List of bus taps with their position, its old bit and the new bit to be assigned This parameter is required.

Examples

```
#command to change bit from 50 to 49
changeBitNumber -pg @worklib.teste_1(tbl_1):page(1) -taps [list 2900 8900 {50} {49}]
```

changeRefdes

It replaces the reference designator (RefDes) of the specified components with the user-assigned reference designator or the tool-defined reference designator. If 'toolDef' is used then <user_refdes> will not be honored.

Return Type

INT

Syntax

```
changeRefdes -userDef|-toolDef [list {<spath_of_component>} <user_refdes>]
```

Parameters

Parameter	Type	Description
`userDef`	`STRING`	Use this keyword to assign user-defined reference designator values. This parameter is required.
`toolDef`	`STRING`	Use this keyword to use System Capture generated reference designator values. This parameter is required.

Examples

```
#command to assign user-defined reference designator, 'M2' to the
#specified component.
changeRefdes -userDef [list {"@worklib.alpha(tbl_1):\I1\"} M2
#command to assign System Capture generated reference designator
#to the specified component.
changeRefdes -toolDef [list {"@worklib.alpha(tbl_1):\I1\"}]
```

Related Commands

reassignBlockRefdes

closeItem

It closes the specified page if opened in same session. Page path can be found using sch::dbGetActivePageSpath when the same page is opened and active. If the wrong page path is specified, then it will return 1 for failure, otherwise 0 for success.

Return Type

INT

Syntax

```
closeItem <item_id>
```

Parameters

Parameter	Type	Description
`item_Id`	`STRING`	Path of the page to be closed This parameter is required.

Examples

```
#page path is in format of @lib.block(view):page(number)
closeItem @worklib.workshop1(tbl_1):page(5)
#it will close page(5) of workshop block if it exists and is opened in same session.
```

Related Commands

dbGetActivePageSpath

openItem

closeProject

Closes the current project. returns 1 on success.

Return Type

BOOL

Syntax

```
sch::closeProject
```

Examples

```
sch::closeProject
```

Related Commands

createProject

openProject

cnsGetBusMemberXNet

Returns a list of member XNets of the specified bus in the design.

Return Type

LIST

Syntax

```
cnsGetBusMemberXNet <busID> ?design_name?
```

Parameters

Parameter	Type	Description
`busID`	`CNSOBJID`	The ID of specified bus object. This parameter is required.
`design_name`	`STRING`	The name of the design enclosed within straight double quotes (" "). If this value is not specified, by default, the root design name is used. This parameter is optional.

Examples

```
#Command to list member xnets of a bus with ID 'dbc:0x00000004'.

cnsGetBusMemberXNet dbc:0x00000004
```

cnsGetName

Use cnsGetName to get the name of the object with specified ID.

Return Type

STRING

Syntax

```
cnsGetName <objId>
```

Parameters

Parameter	Type	Description
`objId`	`CNSOBJID`	Unique Object Identifier for the constraint object This parameter is required.

Examples

```
set listNets [ cnsGetNetInDesign "test_design"]
#dbc:0x00000001 dbc:0x00000002 dbc:0x00000003 dbc:0x00000004 dbc:0x00000005
cnsGetName [lindex $listNets 0]
#output - NET1
cnsGetName [lindex $listNets 1]
#output - NET2
```

Related Commands

cnsGetKey

cnsGetNetInDesign

cnsGetXNetInDesign

collapseAll

Recursively closes the hierarchical tree view of the pages and other items in the currently selected block in the Project explorer window.

Return Type

NONE

Syntax

```
collapseAll
```

Examples

```
collapseAll
```

Related Commands

collapseThisGroup

collapseTree

collapseThisGroup

Closes the hierarchical tree view of the currently selected group in the Project explorer window.

Return Type

NONE

Syntax

```
collapseThisGroup
```

Examples

```
collapseThisGroup
```

Related Commands

collapseAll

collapseTree

collapseTree

Closes the entire selected hierarchical tree view of the design displayed in the Project Explorer window.

Return Type

NONE

Syntax

```
collapseTree
```

Examples

```
collapseTree
```

Related Commands

collapseAll

collapseThisGroup

compile

Compiles the block

Return Type

NONE

Syntax

```
sch::compile <sBlockName>
```

Parameters

Parameter	Type	Description
`<sBlockName>`	`STRING`	Name of the block This parameter is required.

Examples

```
sch::compile workshop1
```

convertToCache

Converts a standard Allegro System Capture project to a cache-enabled project

Return Type

INT

Syntax

```
convertToCache
```

Examples

```
convertToCache
```

copyBlockAs

Use copyBlockAs to create a new copy of an existing block with the provided name. If the name has spaces or special characters, enclose the name in curly braces {block name}.

Return Type

BOOL

Syntax

```
sch::copyBlockAs <currentBlockName> <newBlockName>
```

Parameters

Parameter	Type	Description
`currentBlockName`	`STRING`	The name of the source block to be copied. If the name has spaces enclose name in curly braces {} This parameter is required.
`newBlockName`	`STRING`	The name for the new block name to be created. If the name has spaces enclose name in curly braces {} This parameter is required.

Examples

```
#command to copy a block named "mid" to new
#block called "mid copy"
sch::copyBlockAs {mid} {mid copy}
```

copyPage

It copies the current page data to the clipboard. Paste page command is used after this command to paste the copied page.

Return Type

BOOL

Syntax

```
copypage <pagePath>
```

Parameters

Parameter	Type	Description
`pagePath`	`STRING`	Path of the page to be copied in lcv format, that is, the path for page 1 of test block: @worklib.test(tbl_1):page(1) This parameter is required.

Examples

```
copy @worklib.workshop1(tbl_1):page(4)
```

Related Commands

pasteAfterCurrentPage

pasteBeforeCurrentPage

copyProjectAs

Makes a new copy of project with a new name and root design name.

Return Type

BOOL

Syntax

```
copyProjectAs <source cpm file path> <new project foolder> <new project name> <new
design name>
```

Parameters

Parameter	Type	Description
`sourceCpmFilePath`	`STRING`	Full path of the source cpm file to be copied This parameter is required.
`newProjectFolder`	`STRING`	Location where the new project will be created This parameter is required.
`newProjectName`	`STRING`	Name for the new project This parameter is required.
`newDesignName`	`STRING`	Name for the new root design This parameter is required.

Examples

```
copyProjectAs "{D:/test_data/test/test.cpm} {D://test_data/test23} test23.cpm
test_copy"
```

createBlock

It creates a new block with the name specified in the current project. The new block is added to the design library.

Return Type

INT

Syntax

```
createBlock <block_name>
```

Parameters

Parameter	Type	Description
`block_name`	`STRING`	name of the block to be created This parameter is required.

Examples

```
#command to create new block with name "new_block"
createBlock {new_block}
```

Related Commands

copyBlockAs

createDiffPair

It creates a differential pair for the two selected nets.

Return Type

BOOLEAN

Syntax

```
createDiffPair <diffpair_name>
```

Parameters

Parameter	Type	Description
`diffpair_name`	`STRING`	Name of the differential pair object to be created This parameter is required.

Examples

```
#for the two selected nets, create a differential pair with
#name "DP_DDR_CLK"
selectObject -occPath
{@worklib.source(tbl_1):\I18\@worklib.mid(tbl_1):\I1\@worklib.low(tbl_1):page(1)} -type
ROUTE 9870 9085
selectObject -occPath
{@worklib.source(tbl_1):\I18\@worklib.mid(tbl_1):\I1\@worklib.low(tbl_1):page(1)} -type
ROUTE + 9879 9298
createDiffPair "DP_DDR_CLK"
```

Related Commands

cnsGetDiffPairConstraint

cnsAutoCreateDiffPair

cnsGetDiffPairInDesign

cnsGetDiffPairMemberNet

cnsGetDiffPairMemberXNet

cnsGetDiffPairParentClass

createHybrid

It creates a raw hybrid component to host hybrid content. It returns a new hybrid handle associated with the newly created hybrid components. The handle can be further used to refer to this component.

Return Type

INT

Syntax

```
sdaUI::createHybrid <name> <url>
```

Parameters

Parameter	Type	Description
`name`	`STRING`	name of the hybrid component This parameter is required.
`url`	`STRING`	URL of the content to host This parameter is required.

Examples

```
sdaUI::createHybrid {myblog} {http://myblog.com}
# command will return hybrid handle
# command will create hybrid component to host contents of myblog.com
```

Related Commands

hybridDialog

hybridTab

openURLDialog

openURLTab

createHybridDialog

Creates a hybrid dialog for already created hybrid components. It returns the handle of the newly created dialog. The handle can be used to control visibility of the dialog.

Return Type

INT

Syntax

```
sdaUI::createHybridDialog <hybridHandle> <name> <title> ?icon? ?pos? ?size? ?isModal? ?
isResizable?
```

Parameters

Parameter	Type	Description
`hybridHandle`	`INT`	Handle of already created hybrid component This parameter is required.
`name`	`STRING`	Name of hybrid dialog This parameter is required.
`title`	`STRING`	Title of hybrid dialog This parameter is required.
`icon`	`STRING`	The path to the icon file. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32 and 48x48. This parameter is optional. Default value is `{ }`.
`pos`	`LIST`	top-left position where to display the dialog This parameter is optional. Default value is `{0 0}`.
`size`	`LIST`	Size of hybrid dialog This parameter is optional. Default value is `{800 800}`.
`isModal`	`BOOL`	Option to create modal dialog This parameter is optional. Default value is `1`.
`isResizable`	`BOOL`	Option to create re-sizable dialog This parameter is optional.

Examples

```
sdaUI::createHybrid {myblog} {http://myblog.com}
#output: 59
sdaUI::createHybridDialog 59 {blogdialog} {This is my blog}
#output: 62
sdaUI::setVisibility 62 1
```

Related Commands

createHybrid

setVisibility

createSchematicPage

It creates a page in the given block and opens it. The page is added to the end of the list of pages of that block.

Return Type

INT

Syntax

```
createSchematicPage <blockName>
```

Parameters

Parameter	Type	Description
`blockName`	`STRING`	block in which the page will be created This parameter is required.

Examples

```
createSchematicPage labrun
#If the block "labrun" contains four pages,
#executing this command will create a fifth page,
#named page(5).
```

Related Commands

deleteItem

crossprobeInTopology

Highlights a component from the Power Topology dashboard in the Power Topology Flat View.

Return Type

NONE

Syntax

```
sdaReliability::crossprobeInTopology <spath> <RailName> <Refdes> <flag>
```

Parameters

Parameter	Type	Description
`spath`	`STRING`	spath of the component to be highlighted This parameter is required.
`railName`	`STRING`	Name of the Power Rail in which the component is to be highlighted/dehigl This parameter is required.
`Refdes`	`STRING`	Refdes of the component This parameter is required.
`flag`	`INT`	Flag to indicate whether to highlight or dehighlight the component This parameter is required.

Examples

```
sdaReliability::crossprobeInTopology "@worklib.ref_5g(tbl_1):\\I81\\:\\M326\\"
"VPH_PWR" "U6" 0
#Command will highlight, indicated by 0, U6 in Power Rail VPH_PWR using the spath of
the component.
```

dbAreSameOccProperties

It returns 1 if the first occurrence property and the second occurrence property point to the same master property, otherwise 0. This is useful for reducing the redundant computation.

Return Type

BOOL

Syntax

```
sch::dbAreSameOccProperties <prop1Id> <prop2Id>
```

Parameters

Parameter	Type	Description
`<prop1Id>`	`DBID`	dbID of the first property This parameter is required.
`<prop2Id>`	`DBID`	dbID of the second property This parameter is required.

Examples

```
#command to check whether two occurrence properties identified by the DBID are same
sch::dbAreSameOccProperties db:00000017 db:00000018
#output: 0
```

Related Commands

dbGetPropNameVal

dbGetProperties

dbGetBaseNetAttrState

Returns the status of any route whether it is base net or not. Return value 1 indicates base net and 0 indicates non-base net.

Return Type

STRING

Syntax

```
sch::dbGetBaseNetAttrState <component_spath>
```

Parameters

Parameter	Type	Description
`component_spath`	`STRING`	spath of Net whose base status is required. This parameter is required.

Examples

```
#Sample code to get the base net status:
sch::dbGetBaseNetAttrState [ sch::dbxGetSPath [ sch::dbxGetSelectedItems [
sch::dbxGetActivePage ] ] ]


# Sample output :
1
#1 indicates base net and 0 indicates non-base net.
```

dbGetInfoFromInstanceSpath

Returns the X and Y coordinates and page spath for the given component spath. The results will be returned in JSON format which is processed further as required.

Return Type

STRING

Syntax

```
sch::dbGetInfoFromInstanceSpath <component_spath>
```

Parameters

Parameter	Type	Description
`component_spath`	`STRING`	spath of component whose X,Y coordinates and page spath are required. This parameter is required.

Examples

```
#Sample code to get page info In command line with the given component's spath:

package require schPageUtils

set instSpath "@worklib.syscap_dehdltitleblock(tbl_1):\\I1\\"
set instanceJSON [sch::dbGetInfoFromInstanceSpath $instSpath]
if {$instanceJSON != {}} {
set instDict [ json::json2dict $instanceJSON]
set pageSpath [dict get $instDict id]
set xpos [dict get $instDict x]
set ypos [dict get $instDict y]
puts "X,Y coordinates: $xpos, $ypos"
set pageList [schPageUtils::getIndexList id $pageSpath]
set x [schPageUtils::getDesignPagesAllInfo]
foreach index $pageList {
puts "Page Name: [schPageUtils::getPageName $x $index]"
puts "Page Id : [schPageUtils::getPageId $x $index]"
}
}


# Sample output :

# X,Y coordinates: 4550, 4300
# Page Name: Page
# Page Id : @worklib.mydes(tbl_1):Page(1)
```

dbGetLibCellViewBBox

It returns the coordinates for the top-left and bottom-right corners of the bounding box for the instance specified by the library, cell, and view. If the component specified by the lib:cell:view is not instantiated in the design, then it returns '{0 0} {0 0}'. The same value is returned if the parameter is invalid.

Return Type

BBOX

Syntax

```
sch::dbGetLibCellViewBBox <library> <cell> <view>
```

Parameters

Parameter	Type	Description
`library`	`STRING`	library name of instance This parameter is required.
`cell`	`STRING`	cell name of instance This parameter is required.
`view`	`STRING`	view name of instance This parameter is required.

Examples

```
sch::dbGetLibCellViewBBox discrete cap sym_1
#Output: {-38100 0} {38100 25400}
```

Related Commands

dbGetLibCellView

dbGetPropertiesFromSPath

Returns the list of all properties in name and value pairs for the given spath.

Return Type

LIST

Syntax

```
dbGetPropertiesFromSPath <spath>
```

Parameters

Parameter	Type	Description
`spath`	`STRING`	spath of object whose properties are required This parameter is required.

Examples

```
#sample code to get properties for component.

sch::dbGetPropertiesFromSPath [sch::dbGetSPath [ sch::dbGetSelectedItems [
sch::dbGetActivePage ]]]

# Output -
# {CDS_LIBRARY_ID 7408_TSSOP-HF-74LVC08} {CDS_LIBRARY_PHYSICAL_ID 7408_TSSOP-HF-
74LVC08} {CDS_LMAN_SYM_OUTLINE 0,0,304800,635000}
# {CDS_PART_NAME 7408_TSSOP-HF-74LVC08} {CHIPS_PART_NAME 7408} {HAS_FIXED_SIZE 4B}
{LOCATION U1} {PACK_TYPE TSSOP-HF} {PART_NAME
# 7408} {VALUE 74LVC08}

# sample code to get properties for pin.

sch::dbGetPropertiesFromSPath [sch::dbGetSPath [ sch::dbGetSelectedItems [
sch::dbGetActivePage ]]]

# sample output
# {PN 11} {SEC 4}

# sample code to get properties for net.

sch::dbGetPropertiesFromSPath [sch::dbGetSPath [ sch::dbGetSelectedItems [
sch::dbGetActivePage ]]]

# sample output
# {PHYS_NET_NAME ABC}
```

dbGetPropertyValueFromSPath

Returns the property value for the specified object spath and property name.

Return Type

STRING

Syntax

```
dbGetPropertyValueFromSPath <spath> <prop_name>
```

Parameters

Parameter	Type	Description
`spath`	`STRING`	spath of the object This parameter is required.
`prop_name`	`STRING`	property name whose value is required This parameter is required.

Examples

```
#sample code to get PNN value of a net.

sch::dbGetPropertyValueFromSPath [sch::dbGetSPath [ sch::dbGetSelectedItems [
sch::dbGetActivePage ]]] {PHYS_NET_NAME}

#sample output
# ABC

#sample code to get LOCATION (refdes) for a comp.

sch::dbGetPropertyValueFromSPath [sch::dbGetSPath [ sch::dbGetSelectedItems [
sch::dbGetActivePage ]]] {LOCATION}

#sample output
# U1
```

Related Commands

dbGetActivePage

dbGetSelectedItems

dbGetSPath

dbxGetRefDesAndOriginFromInstSpath

Returns the refdes and its origin for the given component spath. The results will be returned as array contains refdes and origin.

Return Type

STRING

Syntax

```
sch::dbxGetRefDesAndOriginFromInstSpath <component_spath>
```

Parameters

Parameter	Type	Description
`component_spath`	`STRING`	spath of component whose redes and origin required. This parameter is required.

Examples

```
#Sample code to get refdes and it's origin.
array set test [ join [ sch::dbxGetRefDesAndOriginFromInstSpath [ sch::dbGetSPath [
sch::dbGetSelectedItems [ sch::dbGetActivePage ]]]]]
puts $test(refdes)
puts $test(origin)


# Sample output :
RN1 #Represents refdes
32 #represents origin
```

delCustomVar

It deletes the specified custom variable defined between START_CUSTOMVAR and END_CUSTOMVAR from project cpm.

Return Type

BOOLEAN

Syntax

```
delCustomVar -name <variable_name>
```

Parameters

Parameter	Type	Description
`-name <variable_name>`	`STRING`	name of the variable to be deleted This parameter is required.

Examples

```
#add the custom directive named "BETA" with value 123
#and defined type is the project Type
addCustomVar -name BETA -value "123" -type "Project Defined"
#now delete the variable BETA
delCustomVar -name BETA
```

Related Commands

addCustomVar

modifyCustomVar

delete

It deletes the selected objects on a canvas. Objects can be selected using the selectObject command. Running this command without any objects selected will have no effect.

Return Type

NONE

Syntax

```
delete
```

Examples

```
#in the following example, a block is added to page(1)
#of the "labrun" block, selected, and deleted.
addBlock Rect -pg @worklib.labrun(tbl_1):page(1) -rect [list 250 250] [list 1250 1250]
selectObject -occPath @worklib.labrun(tbl_1):page(1) -type GRAPHIC_BLOCK 750 750
delete
```

Related Commands

selectObject

deleteItem

deleteAllAttachments

It deletes all the attachment files that are attached to the currently opened design. It returns a string specifying the status whether the command ran successfully or not. It creates a backup of the design before deleting the attachments from the design. The backup path of the design is specified in the return status.

Return Type

STRING

Syntax

```
sch::deleteAllAttachments
```

Examples

```
sch::deleteAllAttachments
#on successful deletions of attachement:
#the contents of the attachment have been successfully removed.
#backup of the old System Capture database created at:
#D:/testcases/174/testProj/top/logic/top.sdax.1565685635.bak
#when no attachement are present:
#there are no attachment present in System Capture database.
```

Related Commands

deleteAttachment

getAllAttachments

getAttachment

listAttachments

storeAttachment

deleteAttachment

It deletes the specified attachment file in the currently opened design. On successful completion, it indicates that the file is deleted successfully. It also provides path of the backup of the design before the command was executed. On failure, it reports the error of the missing attachment file in the currently opened design. The attachment file names can be queried using the sch::listAttachments command.

Return Type

STRING

Syntax

```
sch::deleteAttachment <attachment_filename>
```

Parameters

Parameter	Type	Description
`attachment_filename`	`STRING`	specifies the attachment file that will be removed from the currently opened design. This parameter is required.

Examples

```
#the following command deletes the attachment "test.png"
#from the currently opened design
sch::deleteAttachment test.png
#1 file(s) successfully deleted from System Capture database.
#backup of the old System Capture database created at:
#D:/testcases/174/testProj/top/logic/top.sdax.1565690332.bak
#the following command indicates an error on missing attachment
#file "top2.cpm" from the current opened design
sch::deleteAttachment top2.cpm
#Error: Missing file. The following file 'top2.cpm' could not be
#deleted from the System Capture database.
```

Related Commands

deleteAllAttachments

getAllAttachments

getAttachment

listAttachments

storeAttachment

deleteDiffPair

It deletes an existing differential pair.

Return Type

NONE

Syntax

```
deleteDiffPair [list <name_of_diffpair>]
```

Parameters

Parameter	Type	Description
`[list <name_of_diffpair>]`	`STRING`	Specify list of differential pair names to be deleted This parameter is required.

Examples

```
#command to delete a differential pair with name "DP_DDR_CLK"
deleteDiffPair [list "DP_DDR_CLK"]
```

Related Commands

createDiffPair

cnsGetDiffPairConstraint

cnsAutoCreateDiffPair

cnsGetDiffPairInDesign

cnsGetDiffPairMemberNet

cnsGetDiffPairMemberXNet

cnsGetDiffPairParentClass

deleteDirective

It deletes a project directive and returns true if the directive is deleted successfully.

Return Type

BOOL

Syntax

```
cps::deleteDirective <sectionName> <directiveName>
```

Parameters

Parameter	Type	Description
`sectionName`	`STRING`	Name of the section in which the directive to delete is present This parameter is required.
`directiveName`	`STRING`	Name of the directive to delete This parameter is required.

Examples

```
#command to delete the project directive JUNCTION_SIZE from the GLOBAL section.
cps::deleteDirective "GLOBAL" "JUNCTION_SIZE"
```

Related Commands

addDirectiveValue

deleteItem

It deletes the page provided in the -pg parameter. Command without parameter will delete the currently selected page.

Return Type

BOOL

Syntax

```
deleteItem ?-pg <page_name>?
```

Parameters

Parameter	Type	Description
`-pg <page_name>`	`STRING`	name and address of the page to be deleted This parameter is optional. Default value is `current page`.

Examples

```
#command to delete page 1 of copy block
deleteItem @worklib.mid_copy(tbl_1):page(1)
#command to confirm and delete currently selected page
deleteItem
```

deleteProp

It deletes the property from the property list of the selected component.

Return Type

NONE

Syntax

```
deleteProp -name <value>
```

Parameters

Parameter	Type	Description
`-name`	`STRING`	Name of the property to be deleted from the selected object. This parameter is required.
`-value`	`STRING`	The value of the property. This parameter is optional.
`-scope {value}`	`STRING`	The scope of the search for the specified object, the entire design or the current block. This parameter is optional.
`-scope2 {value}`	`STRING`	Where to search for the specified object, in components or nets This parameter is optional.
`-mode`	`INT`	Type of the match which is wildcard This parameter is optional.
`-regex`	`NONE`	Use regex for matching This parameter is optional.
`-strict`	`NONE`	Use strict matching This parameter is optional.

Examples

```
deleteProp -name "7U7"
deleteProp -scope "Entire Design" -scope2 "Net" -name "7U7" -value "*" -mode 1
```

deleteSchematicPage

Deletes a schematic page from a specific block.

Return Type

INT

Syntax

```
deleteSchematicPage <block_name> <page_name>
```

Parameters

Parameter	Type	Description
`block_name`	`STRING`	The name of the block. This parameter is required.
`page_name`	`STRING`	The name of the page to be deleted. This parameter is required.

Examples

```
#command to delete page 4 from block1
deleteSchematicPage block1 page(4)
```

descend

It works on the selected block instance. Select the block instance placed on schematic and trigger the descend command. Page 1 of the corresponding will be opened in occurrence of the selected instance.

Return Type

NONE

Syntax

```
descend
```

Examples

```
#command to select any block instance
descend
#Page 1 of block will be opened in selected instance occurrence.
```

Related Commands

ascend

displayBitNumbers

It changes the visibility of a bit number of bus taps on a selected bus.

Return Type

INT

Syntax

```
sch::displayBitNumbers <display option>
```

Parameters

Parameter	Type	Description
`<display option>`	`STRING`	Show or hide bit numbers. This parameter is required. Default value is `NONE`.

Examples

```
sch::displayBitNumbers ON
#makes visible bit numbers of bus taps connected on the selected bus
sch::displayBitNumbers OFF
#hides bit numbers of bus taps connected on the selected bus
```

displayName

It toggles the visibility of the route name on the canvas.

Return Type

NONE

Syntax

```
sch::displayName
```

Examples

```
sch::displayName
#output: annotate -pg @worklib.workshop1(tbl_1):page(4) ON
```

dumpMenus

Writes all menu names and their respective properties to a specified text file in the JSON format.

Return Type

STRING

Syntax

```
cps::dumpMenus <file_path>
```

Parameters

Parameter	Type	Description
`file_path`	`STRING`	File where the menus will be written This parameter is required.

Examples

```
cps::dumpMenus {C:/scripts/dumpmenus.txt}
```

dumpSearchResults

Dump search results to an external file.

Return Type

NONE

Syntax

```
cps::dumpSearchResults <path> <file_name_suffix>
```

Parameters

Parameter	Type	Description
`path`	`STRING`	Path of the file This parameter is required.
`file_name_suffix`	`STRING`	suffix which gets appended to the file This parameter is required.

Examples

```
cps::dumpSearchResults "D:/test" "_test"
#Here path "D:/test" should exist.
#Suffix "_test" will get appended to the files.
```

dumpViolations

It saves the violations reported in the Violation window in a file.

Return Type

NONE

Syntax

```
dumpViolations <file_path>
```

Parameters

Parameter	Type	Description
`file_path`	`STRING`	The file name or full path where the violation data is saved. This parameter is required.

Examples

```
dumpViolations {/home/user1/data/violationFile.txt}
```

editPackagingOptions

Edit packaging options for selected block instance like Reference Designator assignment options in design hierarchy, Global Net Aliasing options, Reuse Instance Name and Block Instance Name

Edit Packaging Options can also used to rename the block instance or Hierarchical block instance.

Return Type

INT

Syntax

```
editPackagingOptions <?-o? | ?-range [list <range_start> <range_end>] ? | ?-s
<block_suffix>? | ?-p <block_prefix>? | -offset <offset_num> | -pattern [list
USER_STRING=<str_option> USER_SUFFIX=<str_suffix_option> ]> [-g [list <global_sig_orig>
<global_sig_new>] [-reuse <reuse_instance_name>] -name <block_inst_name> | -name
<block_inst_name> -spath <Inst_spath> -all
```

Parameters

Parameter	Type	Description
`-o`	`INT`	Edit packaging option use to the Optimized mode. This parameter is optional.
`-range [list <range_start> <range_end>]`	`LIST`	Edit block packaging option to use the Range mode. The range_start and range_end values specify the start and end value of the range This parameter is optional.
`-s <block_suffix>`	`STRING`	Edit block packaging option to use the Suffix mode with the specified suffix value. This parameter is optional.
`-p <block_prefix>`	`STRING`	Edit block packaging option to use the Prefix mode with the specified prefix value. This parameter is optional.
`-g [list <global_sig_orig> <global_sig_new>`	`LIST`	Edit global aliasing for the selected block. This parameter is optional.
`-reuse <reuse_instance_name>`	`STRING`	Edit/Specify reuse instance name on the selected block This parameter is optional.
`-name`	`STRING`	Block Instance Name This parameter is required.
`-spath`	`STRING`	Spath of Instance Name This parameter is required.
`-all`	`STRING`	If User wants to rename Hierarchical Block This parameter is optional.

Examples

```
#Command to use optimized block packaging option
#for selected block instance
editPackagingOptions -o

#Apply suffix option SS and apply global aliasing of net VCC_TOP
#in current design to global net VCC_MID of block mid
editPackagingOptions -s SMID -g [list VCC_MID,VCC_TOP]

#Reset global aliasing options for all nets in the low block
editPackagingOptions -g reset

#Apply range option for specifying the Reference designator
#number range from 50 to 100
editPackagingOptions -range [list 50 100]

#Apply offset on master reference designator assignments
#to adjust numbers by offset 100
editPackagingOptions -offset 100

# Rename the selected "mid" Block Instance as mid_ren
editPackagingOptions -name mid_ren

#rename mid block instance(if spath available) as mid_ren
editPackagingOptions -name mid_ren -spath ""@worklib.mlb(tbl_1):\I1\"

#rename mid hierarchical block instance as mid_ren
editPackagingOptions -name mid_ren -spath ""@worklib.mlb(tbl_1):\I1\" -all
```

electricalStressPreferences

Opens 'Electrical Stress Settings' dialog box.

Return Type

NONE

Syntax

```
sdaReliability::electricalStressPreferences
```

Examples

```
sdaReliability::electricalStressPreferences
```

execute

Helps in calling JavaScript API. The JavaScript function which needs to be called must be used in creating the component whose handle has been passed.

Return Type

NONE

Syntax

```
cps::execute <componentHandle> <JavaScriptFunctionName>
```

Parameters

Parameter	Type	Description
`<componentHandle>`	`INT`	Component handle of the widget whose Javascript API needs to be called This parameter is required.
`<JavaScriptFunctionName>`	`STRING`	JavaScript function and arguments that need to be called from Tcl This parameter is required.

Examples

```
# Following command will return the component handle of the mentioned widget. Component
name needs to passed as argument.
set compHandle [cps::findComponentByName PAGE_SUMMARY_GUI]
puts $compHandle
# An integer value as component handle is returned.

# This command will call the JavaScript API passed as second argument for the component
whose handle has been passed as first argument below
cps::execute $compHandle { javaScriptDummyFunction () }
```

expandAll

Fully expands the selected node in Project explorer tree. This means that all children of the node are now made visible. This command does not take any parameters.

Return Type

NONE

Syntax

```
expandAll
```

Examples

```
expandAll
```

expandThisGroup

Expands just the selected node in the Project explorer tree.

Return Type

NONE

Syntax

```
expandThisGroup
```

Examples

```
expandThisGroup
```

expandTree

Expands the entire hierarchical tree view of the design displayed in the 'Design Explorer' window.

Return Type

NONE

Syntax

```
expandTree
```

Examples

```
expandTree
```

exportAuditDashboardAsCsv

Generates a CSV file with the schematic audit data. The CSV file name with the full path is specified as the command parameter.

Return Type

NONE

Syntax

```
sdaReliability::exportAuditDashboardAsCsv <file_path>
```

Parameters

Parameter	Type	Description
`file_path`	`STRING`	CSV filename along with the full path to the location where the generated CSV is to be saved This parameter is required.

Examples

```
sdaReliability::exportAuditDashboardAsCsv "../auditCSV.csv"
```

exportAuditDashboardAsPdf

Generates a PDF file with schematic audit data. The PDF location, including the filename, is specified as command parameter.

Return Type

NONE

Syntax

```
sdaReliability::exportAuditDashboardAsPdf <file_path>
```

Parameters

Parameter	Type	Description
`file_path`	`STRING`	path, including the filename, to the the location where the generated PDF file is to be saved This parameter is required.

Examples

```
sdaReliability::exportAuditDashboardAsPdf "../auditPDF.pdf"
```

Related Commands

exportAuditDashboardAsCsv

exportDashboardAsCsv

Generates a CSV file using the dashboard data. The CSV file name with the full path is specified as the command parameter, along with the mode.
Mode specifies the feature for which CSV needs to be generated.
For MTBF, set the mode as 1
For Power Topology, set the mode as 2
for Thermal Analysis, set the mode as 3

Return Type

NONE

Syntax

```
sdaReliability::exportDashboardAsCsv <mode> <file_path>
```

Parameters

Parameter	Type	Description
`mode`	`INT`	Specifies the feature for which a CSV file needs to be created, such as MTBF, Power Topology, and Thermal Analysis. This parameter is required.
`file_path`	`STRING`	Name and location for the CSV to be created This parameter is required.

Examples

```
sdaReliability::exportDashboardAsCsv 1 "../dashbaordDump.csv"
#Command will generate csv file for dashboard data for MTBF Analysis

sdaReliability::exportDashboardAsCsv 2 "../dashbaordDump.csv"
#Command will generate csv file for dashboard data for Power Topology Analysis

sdaReliability::exportDashboardAsCsv 3 "../dashbaordDump.csv"
#Command will generate csv file for dashboard data for Thermal Analysis
```

Related Commands

sdaReliability::runMTBFAnalysis

sdaReliability::extractPTree

sdaReliability::runThermalAnalysis

exportDashboardAsPdf

Generates a PDF file using the dashboard data. The PDF file name with the full path is specified as the command parameter, along with the mode.
Mode specifies the feature for which PDF needs to be generated.
For MTBF, set the mode as 1
For Power Topology, set the mode as 2
for Thermal Analysis, set the mode as 3

Return Type

NONE

Syntax

```
sdaReliability::exportDashboardAsPdf <mode> <file_path>
```

Parameters

Parameter	Type	Description
`mode`	`INT`	Specifies the feature for which a PDF file needs to be created, such as MTBF, Power Topology, and Thermal Analysis. This parameter is required.
`file_path`	`STRING`	Name and location for the PDF to be created This parameter is required.

Examples

```
sdaReliability::exportDashboardAsPdf 1 "../dashboardDump.pdf"
#Command is used for generating PDF file for the dashbaord results for MTBF Analysis

sdaReliability::exportDashboardAsPdf 2 "../dashboardDump.pdf"
#Command is used for generating PDF file for the dashbaord results for Power Topology
Analysis

sdaReliability::exportDashboardAsPdf 3 "../dashboardDump.pdf"
#Command is used for generating PDF file for the dashbaord results for Thermal Analysis
```

Related Commands

sdaReliability::runMTBFAnalysis

sdaReliability::extractPTree

sdaReliability::runThermalAnalysis

exportNetList

It generates a physical netlist

Return Type

NONE

Syntax

```
exportNetList
```

Examples

```
exportNetList
```

Related Commands

exportPhysical

exportPhysical

It generates a physical netlist and PCB layout for the current root design or the selected reuse block.

Return Type

NONE

Syntax

```
exportPhysical pcb genpkg ?updateBoard -I <input_board> -O <output_board> -P
<place_options> -L <launch_option> -showReport? ?-lib <lib>? ?-cell <cell>? ?-
promptComplete?
```

Parameters

Parameter	Type	Description
`pcb`	`STRING`	Keyword used to indicate that the physical layout is for a PCB Board. This parameter is required.
`genpkg`	`STRING`	Keyword used to indicate that the package data is to be generated for the design. This parameter is required.

Examples

```
#command to generate a Physical Netlist for root design
exportPhysical pcb genpkg
#command to generate a Physical Netlist for a reuse design
exportPhysical pcb genpkg -lib worklb -cell reuse_design
#command to generate a PCB Layout for root design
exportPhysical pcb genpkg updateBoard -I testimput.brd -O output.brd -P always -L None
-showReport
```

Related Commands

importPhysical

exportReliabilityDashboardAsCsv

Generates a CSV file containing electrical stress data. Complete file path, including the filename, is specified as command parameter.

Return Type

NONE

Syntax

```
sdaReliability::exportReliabilityDashboardAsCsv <file_path>
```

Parameters

Parameter	Type	Description
`file_path`	`STRING`	path to the location where the generated CSV file is to be saved This parameter is required.

Examples

```
sdaReliability::exportReliabilityDashboardAsCsv "../stressCSV.csv"
```

Related Commands

exportReliabilityDashboardAsPdf

exportReliabilityDashboardAsPdf

Generates a PDF file with electrical stress data. The PDF location, including the filename, is specified as command parameter.

Return Type

NONE

Syntax

```
sdaReliability::exportReliabilityDashboardAsPdf <file_path>
```

Parameters

Parameter	Type	Description
`file_path`	`STRING`	Path, including the filename, to the location where the generated PDF is to be saved This parameter is required.

Examples

```
sdaReliability::exportReliabilityDashboardAsPdf "../stressPDF.pdf"
```

Related Commands

exportReliabilityDashboardAsCsv

extractPTree

This command runs the Power Topology analysis on the design based on the settings specified in the 'Power Topology Settings' dialog.
Use the sdaReliability::PTreePreferences command to access the 'Power Topology Settings' dialog.

Return Type

NONE

Syntax

```
sdaReliability::extractPTree
```

Examples

```
sdaReliability::extractPTree
```

Related Commands

sdaReliability::PTreePreferences

find

It finds an item matching a specified criteria. There are two types of Find: 'Find Text' and 'Global Find'. 'Find Text' is used to find electrical and nonelectrical objects. 'Global Find' is used to find electrical objects and special symbols only.

Return Type

NONE

Syntax

```
find ?globalFind | -<match_type> | -scope <type> | -types <list_of_objects>]| -
findinPropName | match_string | -p {<prop-name=propvalue>} | -exclude <"list of sub-
types which needs to be excluded">?
```

Parameters

Parameter	Type	Description
`match_type`	`STRING`	Specifies the type of comparison, such as strict, regular expression, or wildcard. This parameter is optional. Default value is `Wildcard`.
`[-scope <"type">]`	`STRING`	The scope of the search, such as design, page, block, or selection. This parameter is optional. Default value is `Current Block`.
`[-types <"list_of_objects">]`	`STRING`	List of objects to be searched. The values must be separated by spaces. For example, "components pins wires" will run the search in only these objects. This parameter is optional.
`[-findinPropName]`	`STRING`	The text is searched in the property names as well as the values. This parameter is optional.
`[{match_string}]`	`STRING`	The search string to be found. This parameter is optional.
`[-exclude <"list of sub-types which needs to be excluded">]`	`STRING`	This is an internal command and used to exclude sub-types from the list of types. This option is only applicable for Find Text type of commands. This parameter is optional.

Examples

```
#Find Text examples:
find -types "component note route pin" {*}
find -strict -scope "Current Page" -types "component note route pin" -findinPropName
{rr}
find -regex -scope "Current Page" -types "component route pin" {rr*}
find -scope "Current Page" -types "component" -exclude "special-bodies auto-shapes
block-inst" {*}
find -scope "Current Page" -types "route" -exclude "netgroup" {*}
#Global Find Examples:
find globalFind -scope {{Entire Design}} -l {{hello}} -c {{*}} -v {{*}}
find globalFind -scope {{Entire Design}} -n {{net1}} -mode 1
find globalFind -scope {{Entire Design}} -scope2 "Net" -p {{prop=val}} -mode 1
find globalFind -scope {{page}} -scope2 "Net" -p {{prop=val}} {-pages {1}} -blockname
{new_des1} -strict
```

firstPage

It opens the first page of the currently active block. 0 is returned on success, and an empty list is returned on failure.

Return Type

INT

Syntax

```
sch::firstPage
```

Examples

```
#in the following example, the first page of the current block is opened,
#otherwise a message is printed.
if { [sch::firstPage] == {} } { puts "First page is already open." }
```

Related Commands

lastPage

nextPage

previousPage

formatCRefs

It works on the selected navigation links of a port or offpage body. The navigation links are moved into a matrix of given rows and columns.

Return Type

NONE

Syntax

```
sch::formatCRefs <lX> <lY>
```

Parameters

Parameter	Type	Description
`<lX>`	`INT`	Rows This parameter is required.
`<lY>`	`INT`	Columns This parameter is required.

Examples

```
It will move navigation links in a matrix of given rows and columns.
- Select Navigation Links of any port or offpage and chose Format Nav Links (Rows X
Columns)
```

generatePartPropertiesData

Generates Part Manager data for a set of instances or the entire design. For specific instances, specify a list of spaths and set 'bSelectedParts' to TRUE. To run on the entire design, pass an empty list and set 'bSelectedParts' to FALSE. On successfully generating Part Manager data for the given instances or the entire design, 'CP_OK' is returned

After this, the 'cpb::queryPartPropsFromSpath' command can be used to get the property name values list for specific spaths.

Return Type

CP_STATUS

Syntax

```
cpb::generatePartPropertiesData bSelectedPartsFlag spathVecList
```

Parameters

Parameter	Type	Description
`bSelectedPartsFlag`	`BOOL`	Set to TRUE for selected instances and FALSE for entire design This parameter is required.
`spathVecList`	`LIST`	If bSelectedPartsFlag is set to true, this list contains a list of spaths else it is empty list This parameter is required.

Examples

```
#Get a list of spaths of items selected on canvas and check if they have JEDEC_TYPE
property
set spath [ sch::dbGetSPath [ sch::dbGetSelectedItems [sch::dbGetActivePage ] ] ] ]
set vecList [cps::stdVectorStr]
foreach item $spath { $vecList push $item } {
cpb::generatePartPropertiesData 1 $vecList
}
```

Related Commands

queryPartPropsFromSpath

getActionId

Returns the action ID given the context and name of the action.

Return Type

INT

Syntax

```
getActionId <context> <actionName>
```

Parameters

Parameter	Type	Description
`context`	`STRING`	The context in which this action is supported. For schematic canvas actions, this should be "sch" This parameter is required.
`actionName`	`STRING`	Name of the action This parameter is required.

Examples

```
getActionId sch Undo
# prints 12007
```

Related Commands

getActionName

getActionName

Returns the name of the action associated with action ID in a specific context.

Return Type

STRING

Syntax

```
getActionName <context> <actionId>
```

Parameters

Parameter	Type	Description
`context`	`STRING`	The context in which this action is supported. For schematic canvas actions, this should be "sch" This parameter is required.
`actionId`	`INT`	Action ID This parameter is required.

Examples

```
getActionName sch 12007
#prints &Undo
```

Related Commands

getActionId

getActiveContext

Command returns currently active context of application.

Return Type

STRING

Syntax

```
cps::getActiveContext
```

Examples

```
#If schematic page is opened and active,
#then active context would be sch.
cps::getActiveContext

#Output: sch
```

getActiveSPath

Returns top-level design block's spath of the currently opened project. SPaths are unique string identifiers for blocks, pages, instances, nets, and route objects in a project. These are used as input parameter to Tcl APIs that operate on these objects.

Return Type

SPATH

Syntax

```
cps::getActiveSPath
```

Examples

```
#Command to return the sPath of the top-level design named "labrun"
puts "SPath of the active project is [cps::getActiveSPath]"

#SPath of the active project is @worklib.labrun(tbl_1)
```

Related Commands

getSPathByTabIndex

dbGetBlockSPath

dbGetSPathForActiveTab

dbGetSPath

getAllAttachments

It extracts all the attachment files from the currently open System Capture design to the specified output folder path. The specified folder should exist for the successful completion of the command. This command creates an "attachment" folder in the specified folder path and extracts all the attachment files there.

Return Type

STRING

Syntax

```
sch::getAllAttachments <outputfolder>
```

Parameters

Parameter	Type	Description
`<outputfolder>`	`STRING`	absolute path of the already existing output folder. This parameter is required.

Examples

```
#command to successfully extract all the attachments from
#the currently opened design
sch::getAllAttachments d:/attach
#the contents of the attachment have been successfully
#extracted at D:/attach/attachment
#the following command fails as the folder specified is absent
sch::getAllAttachments d:/myattachements
#Error: The output directory specified d:/myattachements
#does not exist.
```

Related Commands

deleteAllAttachments

deleteAttachment

getAttachment

listAttachments

storeAttachment

getAppBuildNumber

This command gives the information about the current build of the product. It does not require any parameter.

Return Type

STRING

Syntax

```
cps::getAppBuildNumber
```

Examples

```
#Command to get the current build number
cps::getAppBuildNumber
```

getAppTitle

Returns the current application title for System Capture.

Return Type

STRING

Syntax

```
cps::getAppTitle
```

Examples

```
# If the title is "Allegro System Capture"
cps::getAppTitle
# Allegro System Capture
```

Related Commands

setAppTitle

getAppVersion

This command gives the information about current release version of the product. It does not require any parameter.

Return Type

STRING

Syntax

```
cps::getAppVersion
```

Examples

```
#Command to get the current release version
cps::getAppVersion
```

getClickedTreeNode

This command will return the handle of the tree node which is being clicked. The node handle will be returned for the node defined under given tree hierarchy name.

Return Type

Handle

Syntax

```
cps::getClickedTreeNode <rootTreeName>
```

Parameters

Parameter	Type	Description
`<rootTreeName>`	`STRING`	Name of the tree hierarchy This parameter is required.

Examples

```
#Command to store the handle of the tree node in nodeHandle
set nodeHandle [ cps::getClickedTreeNode "CP_PROJECT_EXPLORER" ]
```

getConfiguredLibraryTypes

This command returns the list of library modes set by the Pulse server or the CDS_SYSCAP_NEWPROJ_LIBMODE environment variable.
The possible values for the type of project are as follows:
dehdl - signifies Allegro DE-HDL
capture - signifies OrCAD Capture
unified - signifies Allegro Unified

Return Type

STRING

Syntax

```
getConfiguredLibraryTypes
```

Examples

```
# For example, if the library modes set by the environment variable is dehdl and
capture, the following command returns:
getConfiguredLibraryTypes
# Output - dehdl, capture

# The library mode set by Pulse by default is de-hdl. The following command returns the
respective library mode:
getConfiguredLibraryTypes
#output - dehdl
```

getDEHDLColorNames

Returns the names of all valid DE-HDL colors.

Return Type

LIST

Syntax

```
getDEHDLColorNames
```

Examples

```
getDEHDLColorNames
```

Related Commands

getDEHDLImportColorMapping

setDEHDLImportColorMapping

getDEHDLImportColorMapping

Returns the color code for the specified dehdlColor. The color code is returned in the form of a list of RGB color codes (integers).
For invalid color names, an empty list is returned.

Return Type

LIST

Syntax

```
getDEHDLImportColorMapping <dehdlColor>
```

Parameters

Parameter	Type	Description
dehdlColor	STRING	The color name This parameter is required.

Examples

```
#Command with correct parameters returns color code in RGB color codes.
getDEHDLImportColorMapping RED
{255 0 0}
getDEHDLImportColorMapping green
{0 128 0}
#Invalid color input returns empty list.
getDEHDLImportColorMapping gr
{}
```

Related Commands

setDEHDLImportColorMapping

getDEHDLColorNames

getDirectives

Returns a list of available directives within a project section.

Return Type

LIST OF STRINGS

Syntax

```
cps::getDirectives <section name>
```

Parameters

Parameter	Type	Description
`sectionName`	`TCL LIST OF STRING`	name of a section This parameter is required.

Examples

```
cps::getDirectives TOOLS
#output:
{Allegro Design Authoring Team Design Option} {Allegro Discrete Library to Agilent ADS
Translator} ImportIFF {Design Entry HDL (Read Only)} Part_Manager Archiver Archopen
Crefer Rules_Checker DesMgr SigXplore EDIF300 Simulate PICHDLImport PICNetlister
PlaceAndRoute BuildPhysical VerifyPNR VerifySynthesis VariantEditor ExportXML
ExportXMLFromCapture ExportCapture VerifyAllegro VerifyPackaging VerifySimulation
VerifyCheckplus VerifyTemplate PadStackEditor AllegroSymbolEditor ViewAllegroSymbol
ViewConceptSymbol ViewCaptureSymbol EditSimulationViews Pdv ImportXML
ImportXMLToCapture ImportCapture LibExp SearchPart SetupTemplates PowerTree LibCreator
```

getDirectiveValue

It queries the value of a directive defined in the specified section.

Return Type

STRING

Syntax

```
cps::getDirectiveValue <sectionName> <directiveName>
```

Parameters

Parameter	Type	Description
`sectionName`	`STRING`	Name of the section in which directive exists. The supported values are valid section names. This parameter is required.
`directiveName`	`STRING`	Name of the directive This parameter is required.

Examples

```
set location [cps::getDirectiveValue GLOBAL design_loc]
puts $location
#output: './logic'
```

getPageId

It takes ‘pageInfo’ and index into ‘pageInfo’ as argument. It returns spath of page. pageInfo returned from getDesignPagesAllInfo command is used as input.
Range of index is from 1 to total number of pages in design.

Return Type

Spath

Syntax

```
schPageUtils::getPageId pageInfo index
```

getParentTreeNode

This command will return the handle of the parent node of the given tree node in the hierarchy.

Return Type

Handle

Syntax

```
cps::getParentTreeNode <treeNode>
```

Parameters

Parameter	Type	Description
`<treeNode>`	`Handle`	Node handler on which icon needs to be set This parameter is required.

Examples

```
#Command to store the tree handler in the variable called nProjectExplorerParentNode
#for CP_PROJECT_EXPLORER and then create a node named - PSpiceDesign and store
#the handle of this node into variable called designNode

variable nProjectExplorerParentNode ""
set nProjectExplorerParentNode [ cps::getTree CP_PROJECT_EXPLORER ]

set designNode [ cps::createTreeNode $nProjectExplorerParentNode [ cps::getTreeRootData
$nProjectExplorerParentNode ] "" PSpiceDesign 0 ]

#Create the child node of the designNode
set childNodePtr [ cps::createTreeNode $nProjectExplorerParentNode $designNode ""
ChildNode 0 ]

#Now , get the parent tree node which will be same as designNode
set parentNode [ cps::getParentTreeNode $childNodePtr ]
```

Related Commands

createTreeNode

getProjectInfo

It returns the name of the root design and library for a given cpm file.

Return Type

LIST OF STRINGS

Syntax

```
cps::getProjectInfo <path to project file>
```

Parameters

Parameter	Type	Description
`<path to project file>`	`STRING`	Absolute path of a design file This parameter is required.

Examples

```
puts [cps::getProjectInfo /home/user/projects/project00/a.cpm]
```

getResourceFullPath

Returns the full path of any resource item passed. The parameter value (<resourceItemPath>) is the path to any item in the <Cadence_installation>/share/cdssetup/ folder.

The sequence in which the search is performed to find the full path of the resource item is:
Project directory > Home directory > CDS_SITE directory > Cadence installation directory.

The location where the resource item is found is returned as the path of the resource item.

Return Type

STRING

Syntax

```
cps::getResourceFullPath <resourceItemPath>
```

Parameters

Parameter	Type	Description
`resourceItemPath`	`STRING`	Path of the any item available inside <Cadence_installation>/share/cdssetup/ folder. This parameter is required.

Examples

```
# This command searches for the mentioned resource item inside the cdssetup folder and
returns its full path.
cps::getResourceFullPath {common/themes/dark/svgicons/Add.svg}
# Following is the return value as full path of the Add.svg file
# /home/jdoe/Cadence/share/cdssetup/common/themes/dark/svgicons/Add.svg
```

getSDAXPath

Returns the project's sdax database filepath.

Return Type

STRING

Syntax

```
getSDAXPath
```

Examples

```
puts [getSDAXPath]
# returns /home/jdoe/designs/testbed/logic/ref_5g.sdax
```

getSections

Returns a list of available sections from the project loaded in the application.

Return Type

LIST OF STRINGS

Syntax

```
cps::getSections
```

Examples

```
cps::getSections
#Output: TOOLS DESIGN_FLOWS WEB_PAGES PROJECTMGR GLOBAL CONCEPTHDL PKGRXL
#DESIGNSYNC BOMHDL ERCDX CHECKPLUS PDV142 PDV VXL LF INCA_VLOG INCA_VHDL
#NETLIST PIC ACTEL ALTERA MINC SYNTH CREFERHDL ARCHIVER ECSET_MODELS DSSCHGEN
#DESIGNSTUDIO CRM COMPBROWSER CACHE LRM PURGE ADW RFPCB PDF DESIGN FONTS
#SDM FSP VARIANT CANVAS CONSTRAINT_MGR ALLEGRO
```

getVariantNames

Returns a list of variants defined

Return Type

LIST OF STRINGS

Syntax

```
cps::getVariantNames
```

Examples

```
puts [cps::getVariantNames]
```

getXNetDataFromDesign

Writes the details of all the nets and XNets in the design, in a file.

Return Type

STRING

Syntax

```
getXNetDataFromDesign <file_path>
```

Parameters

Parameter	Type	Description
`<file_path>`	`STRING`	path of the file XNetData This parameter is required.

Examples

```
getXNetDataFromDesign {C:\Users\avatans\Desktop\Documentation\xnetData}
```

globalFind

Displays the Global Find dialog box to search for a net, a component, or a property in a design.

Return Type

BOOLEAN

Syntax

```
globalFind -scope "Entire Design|Current Block" -net|-n <net_name> ?[-p [list
<property_name=property_value] ]? globalFind -scope "Entire Design|Current Block" -
lib|-l <library_name> -cell|-c <cell_name> -view|-v <view_name> [-p [list
<property_name=property_value>]] globalFind -scope "Entire Design|Current Block" -
scope2 "Component|Net" -p <property_name=propety_value>
```

Parameters

Parameter	Type	Description
`-scope "Entire Design|Current Block"`	STRING	Defines the scope of the search, whether to search the specified object in the entire design or the current block. This parameter is required.
`-net|-n <net_name>`	STRING	The netname to be searched. For example, -n "alpha". The netname is enclosed within straight double quotes. This parameter is required.
`-p [list <property_name=property_value>]`	STRING	The name and the value of the property to be searched. You can specify a single property or a list of properties. This parameter is optional.
`-lib|-l <library_name>`	STRING	The name of the library in which the component has to be searched. This parameter is required.
`-cell|-c <cell_name>`	STRING	The name of the component to be searched. This parameter is required.
`-view|-v <view_name>`	STRING	The name of the view. This parameter is required.

Examples

```
globalFind -scope {Entire Design} -l {hello} -c {*} -v {*}
```

globalReplace

Replaces instances of an object in the design with another object. You can replace a component, a net, or a property in the design.

Return Type

INT

Syntax

```
globalReplace -scope <Entire Design|block|Page|current_selection> ?-pages <page_no|page
range>? ?-blockname <block_name>? ?match_type? -oldlib <library_name> -oldcell
<cell_name> -oldview <view_name> -newlib <library_name> -newcell <cell_name> -newview
<view_name> [-oldp [list <property_name=propety_value>]] [-key [list
<property_name=property_value>]] ?[-cpath [list <component_spath>]] -preserveUserProp -
preserveRefDes ?-useexistingtr? ?-retainVoltage? | -oldnet <old_net_name> -newnet
<new_net_name> | -scope2 <net|Component> -oldp <property_name=property_value> -newp
<property_name=property_value>|
```

Parameters

Parameter	Type	Description
`-scope <Entire Design|block|Page|current_selection>`	`STRING`	The scope of the search for the specified object - whether to search for the component in the entire design or the current block or pages This parameter is required.
`-oldlib <library_name>`	`STRING`	The name of the existing library to be replaced. This parameter is required.
`-oldcell <cell_name>`	`STRING`	The name of the existing component to be replaced. This parameter is required.

`-scope "Entire Design|Current Block"`	`STRING`	The scope of the search for the specified object - whether to search for the component in the entire design or the current block. This parameter is required.
`-oldp [list <property_name=property_value>]`	`STRING`	The list of properties of the existing component to be replaced. This parameter is optional.
`-oldview <view_name>`	`STRING`	The name of the existing component view to be replaced. This parameter is required.
`-newlib <library_name>`	`STRING`	The new name for the library. This parameter is required.
`-newcell <cell_name>`	`STRING`	The new name for the component. This parameter is required.
`-newview <view_name>`	`STRING`	The new view name for the component. This parameter is required.
`-newp [list <property_name=property_value>]`	`STRING`	The property list of the new component. This parameter is required.
`-k [list <key_prop=value>]`	`STRING`	Key property values assigned to the new component. This parameter is required.
`-i [list <inj_prop=value>]`	`STRING`	Injected property values assigned to the new component. This parameter is required.
`-mapprop [list <prop_name> <prop_value>]`	`STRING`	The properties of the existing component that need to be retained on the new component. This parameter is required.

`unmapprop [list <prop_name> <prop_value>]`	`STRING`	The properties that need to be removed from the new component. This parameter is required.
`-oldnet|-oldn <search net name>`	`STRING`	The name of the net to be replaced. To be enclosed within straight double quotes. This parameter is required.
`-newp [list <property Name=Value>]`	`STRING`	The new properties to be assigned to the new net. This parameter is optional.
`?-pages <page_no|page range>?`	`STRING`	Page no to be assigned if page/block scope is selected This parameter is required.
`?-blockname <block_name>?`	`STRING`	The block name to be assigned if block name is selected This parameter is required.
`?match_type?`	`STRING`	matching type to be assigned if not default This parameter is optional. Default value is `wildcard`.

Examples

```
# In the below command a capacitor component is replaced into another component having
different properties.
# scope is selected as block/page
# page no is provided in field "-pages{1}"
# blockname is also provided in field "-blockname {ptree_testcase}"


globalReplace -oldlib {passives} -oldcell {cap} -oldview {sym_1} -newlib {passives} -
newcell {cap} -newview {sym_1} -preserveUserProp -preserveRefDes -scope {page} -oldp
[list {CDS_LIBRARY_ID=CAP_0603-CDN-CAP-0008,4.7NF,50V,5%=0}
{CDS_LIBRARY_PHYSICAL_ID=CAP_0603-CDN-CAP-0008,4.7nF,50V,5%=0} {PACK_TYPE=0603=1}
{PART_NAME=CAP=0} {PART_NUMBER=CDN-CAP-0008=1} {TOLERANCE=5%=1} {VALUE=4.7nF=1}
{VOLTAGE=50V=1} {PHYS_PAGE=2=-1} {ROT=0=-1} {VER=2=-1} {XY=(-4400,4250)=-1} ] -key
[list {CDS_LIBRARY_ID=CAP_0603-CDN-CAP-0006,2.2UF,10V,40%=0}
{CDS_LIBRARY_PHYSICAL_ID=CAP_0603-CDN-CAP-0006,2.2uF,10V,40%=0} {PACK_TYPE=0603=1}
{PART_NAME=CAP=0} {PART_NUMBER=CDN-CAP-0006=1} {TOLERANCE=40%=1} {VALUE=2.2uF=1}
{VOLTAGE=10V=1} ] -useexistingtr -pages {1} -blockname {ptree_testcase}
```

grid

Changes the grid settings.

Return Type

INT

Syntax

```
grid ?[<display_type>]? -size [list <pin_to_pin_spacing> <elec_grid_spacing>
<doc_grid_spacing>] ?[-style <type>]? -display <value> -unit <type>
```

Parameters

Parameter	Type	Description
`<display_type>`	`STRING`	Type of grid, Electrical or documentation grid. This parameter is optional. Default value is `none`.
`-size [list <pin_to_pin_spacing> <elec_grid_spacing> <doc_grid_spacing>]`	`INTEGER`	Pin-to-pin spacing, electrical grid spacing, and documentation grid spacing. Electric grid spacing and documentation grid spacing are defined as a factor of pin-to-pin spacing. This parameter is required.
`-style <type>`	`STRING`	Style of the grid in terms of dots and lines. This parameter is optional. Default value is `dots`.
`-display <value>`	`INTEGER`	Display electrical grid every <value> grid spacing. This parameter is required.
`-unit <type>`	`STRING`	Unit of the grid definition, Inches or Millimeters. This parameter is required.

hideItems

Hides or displays the list of specified pages. Returns 1 for failure, and 0 for success.

stop_on_error is a provision that allows the command to continue or stop when the operation fails for any one page. When set to 1, the command continues to work on the rest of the pages.

Return Type

ret

Syntax

```
hideItem <item_ids> <item_type> <view_type> <hide_option> <stop_on_error>
```

Parameters

Parameter	Type	Description
`item_ids`	`LIST`	list of page spaths This parameter is required.
`item_type`	`STR`	item type This parameter is required.
`view type`	`STR`	view type This parameter is required.
`hide option`	`INT`	hide option This parameter is required.
`stop_on_error`	`INT`	stop on error or continue in case any error occurs This parameter is required.

Examples

```
hideItems { @worklib.aligntest(tbl_1):Page(1) @worklib.aligntest(tbl_1):Page(2)
@worklib.aligntest(tbl_1):Page(3)} {SCH} {PAGE} {1} {1}
```

Related Commands

hideitem

hideToolBar

It will hide the top most toolbar.

Return Type

NONE

Syntax

```
hideToolBar
```

Examples

```
::hideToolBar
```

Related Commands

showToolbar

importBlock

Imports DE-HDL or System Capture designs as blocks into the current project. For DE-HDL and System Capture designs, the user can import specific blocks.

Return Type

NONE

Syntax

```
importBlock srcProj <source_proj_path> lib <library_name> cell <cell_name> blockOptions
<list[<block_name>:<rw|w>]> ?type <syscap|dehdl>? ?pulseUrl <pulse_design_url>?
```

Parameters

Parameter	Type	Description
`srcProj`	`STRING`	The path to the project file (.cpm or .sdax) from which the block is to be imported. This parameter is required.
`lib`	`STRING`	The name of the library where the cell exists. This parameter is required.
`cell`	`STRING`	The cell or block name to be imported. This parameter is required.
`blockOptions`	`STRING`	The list of blocks to be imported and the mode in which the block is imported. Valid values for the mode are 'r' and 'rw', indicating read-only and read-write modes, respectively. If the block being imported is a hierarchical block, you need to specify the names of child blocks along with the mode in which it is to be imported. This parameter is required.
`type`	`STRING`	The source project type. Valid values are 'syscap' and 'dehdl'. This parameter is optional. Default value is `dehdl`.
`pulseUrl`	`STRING`	The source Pulse URL, used internally when placing parts from Unified Search. Used only for System Capture designs. This parameter is optional.

Examples

```
#command to import the 'tt' block in read-write mode from a System Capture project.
importBlock {srcProj "D:/hier_design.sdax" lib "worklib" cell "tt" blockOptions "tt:rw"
type syscap}
#command to import the 'top' block in read-write mode from a System Capture project.
importBlock {srcProj "../../test123/test123.cpm" lib "worklib" cell "top" blockOptions
"top:rw" type syscap}
#command to import the 'top' block with it's child blocks, in read only mode from
#a System Capture project.
importBlock {srcProj "../../test123/test123.cpm" lib "worklib" cell "top" blockOptions
"top:r,mid:r,low:r" type syscap}
#command to import the 'dtop' block in read-write mode from a DE-HDL project.
importBlock {srcProj "D:/designs/dehdl/dehdl.cpm" lib "dehdl_lib" cell "dtop"
blockOptions "dtop:rw" type dehdl}
#command to import the 'dtop' block in read mode from a DE-HDL project.
importBlock {srcProj "D:/designs/dehdl/dehdl.cpm" lib "dehdl_lib" cell "dtop"
blockOptions "dtop:r"}
#command generated while placing hierarchical block from a shared design from Unified
Search
importBlock {srcProj "../../memory_t/memory_t.cpm" lib "worklib" cell "test_mid"
blockOptions "test_mid:rw" type syscap pulseUrl
pulse://designproject/memory_t:1685960804020/4fce28a4-f5c2-447c-81df-19846cdbb002 }
```

Related Commands

importSheets

reImportBlock

importExternalAllegroBoard

It imports an Allegro Board from an external file

Return Type

BOOL

Syntax

```
importExternalAllegroBoard -board <board_file_path>
```

Parameters

Parameter	Type	Description
`board`	`STRING`	The path to the board file (.brd). This parameter is required.

Examples

```
importExternalAllegroBoard -board "../test123.brd"
```

importLibraryDialog

This command is used to open import library dialog. It works on windows and on Unix it gives an error as "The Import OrCAD Capture libraries functionality is available only on the Windows platform.".

Return Type

none

Syntax

```
importLibraryDialog
```

Examples

```
importLibraryDialog
```

importOLB

This command is used to import library from an olb file.

Return Type

none

Syntax

```
syscapLibDB::importOLB {<olb file path>}
```

Parameters

Parameter	Type	Description
`olb path`	`STRING`	path of olb file This parameter is required.

Examples

```
syscapLibDB::importOLB {D:/Cadence/SPB17.42/tools/capture/library/Amplifier.olb}
```

importPhysical

Imports the physical design data from the Allegro PCB Editor layout database to the logical design.

Return Type

BOOLEAN

Syntax

```
importPhysical <{directory_name}>
```

Parameters

Parameter	Type	Description
`<directory_name>`	`STRING`	name of the directory This parameter is required.

Examples

```
importPhysical "feedback preview {D:\CPMX_PROJECTS\signoise.run} "
```

importPinDelay

It imports the PIN_DELAY data for the pins from the pindelay CSV file.

Return Type

INT

Syntax

```
importPinDelay <file_name>
```

Parameters

Parameter	Type	Description
`file_name`	`STRING`	Specify path of the pin delay (CSV format) to be imported This parameter is required.

Examples

```
#command to import pin delay data from pin delay CSV file
importPinDelay {/home/user1/data/pin_delay_data/pinDelay.csv}
```

Related Commands

importPinDelayFileDialog

importSheets

Imports sheets from a DE-HDL or an Allegro System Capture project to the current project. Additionally, it can be used to replace pages.

Return Type

BOOL

Syntax

```
importSheets srcproj <sourceProjPath> lib <libraryName> cell <cellName> pages <list
[pageNumbers]> ?replacePages <list [pageNumbers]>? ?destPageNum
<destinationPageNumber>? ?beforepage <beforePage>? ?type <sourceProjectType>? ?
preserveRefdes<preserveRefdes>? ?destBlockName<destBlockName>?
```

Parameters

Parameter	Type	Description
`srcproj <sourceProjPath>`	`STRING`	The path to the project file (.cpm or .sdax) from which the sheets has to be imported. This parameter is required.
`lib <libraryName>`	`STRING`	The name of the library where the cell exists. This parameter is required.
`cell <cellName>`	`STRING`	The cell or block name to be imported. This parameter is required.
`pages <list [pageNumbers]>`	`STRING`	The list of page numbers to be imported from the source design. The valid arguments include a list of page numbers and a range of pages like {1:10}. This parameter is required.

`replacePages <list [pageNumbers]>`	`STRING`	The list of page numbers to be replaced from the source design. The valid arguments include a list of page numbers 1,2,3,4 and a range of pages like {1:10} or {1-10}. Portions of the arguments should be specified as range or as single numbers in both the pages parameter replacePages parameter both ways. For example, pages "1-4,5,6" replacePages "4-7,11,12" is valid pages "1-4,5,6" replacePages "4-7,11-12" is not valid pages "1:10" replacePages "10:19" is valid Additionally, the range notation "lower-upper" or "lower:upper" specification for source pages ("pages") is valid only in the replaceSheets commands. This is an optional argument but has no default values, if "replacePages" list is empty this parameter need not be provided. This parameter is optional.
`destPageNum <destinationPageNumber>`	`INT`	The destination page number after which the imported pages are added. Optional in case of replace page. This is an optional argument but has no default value, if "destinationPageNumber" is empty, this parameter need not to be mentioned in command. This parameter is optional.
`beforepage <beforePage>`	`INT`	Specifies whether to add the imported pages before the page, specified by 'destPageNum' parameter. Valid values are '1' and '0', indicating insert pages before page and insert pages after page, respectively. Default value is '0'. This parameter is optional.
`type <sourceProjectType>`	`STRING`	The source project type. Valid values are 'syscap' and 'dehdl'. This parameter is optional. Default value is `dehdl`.

`preserveRefdes<preserveRefdes>`	`INT`	Specifies whether to preserve the packager-assigned reference designators (location property) of instances during import. The user-assigned reference designators will be preserved regardless of this. This parameter is optional. Default value is `1`.
`destBlockName<destBlockName>`	`STRING`	Specify the block name where the sheets has to be imported. This is available only for System Capture projects. Also, this is not mandatory for importSheet command, but mandatory for replace sheet operation from existing System Capture design. Not for DE-HDL to System Capture replace sheet operation. This parameter is optional. Default value is `root block`.

Examples

```
#command to import the sheet 1 from 'dtop' block after page 8,in preserved mode
#from a DE-HDL project.
importSheets {srcProj "D:/designs/dehdl/dehdl.cpm" lib "dehdl_lib" cell "dtop" pages
"1" destPageNum 8 beforepage 0 type "dehdl" preserveRefdes 1}
#command to import the sheet 1 from 'dtop' block after page 8, in non preserved mode
#from a DE-HDL project.
importSheets {srcProj "D:/designs/dehdl/dehdl.cpm" lib "dehdl_lib" cell "dtop" pages
"1" destPageNum 1 beforepage 1 type "dehdl" preserveRefdes 0}
#command to import the sheet 1 from 'src' block of System Capture project after page
#3 of test block.
importSheets {srcProj "D:/designs/n2/logic/n2.sdax" cell "src" pages "1" destPageNum 3
beforepage 0 type "syscap" preserveRefdes 0 destBlockName "test"}
#command to replace sheets 1 to 39 with the sheets 1 to 30 from 'ori' block from
#a System Capture project.
importSheets {srcProj "/export/home/design1/replace_test/copy1/ori.cpm" lib "worklib"
cell "ori" pages "1-39" replacePages "1-39" type "syscap" destBlockName "ori" }
#command to replace sheets 3,4,5,6 with the sheets 10,11,12,13 from 'ori' block from
#a DE-HDL design.
importSheets {srcProj "/export/home/design1/replace_test/copy1/ori.cpm" lib "worklib"
cell "ori" pages "3,4,5,6" replacePages "10,11,12,13" type "dehdl" destBlockName "ori"
}
#command to replace sheets 1 to 4 and 7,8,9 with the sheets 10 to 13, 16,15,14 from
'ori'
#block from a System Capture design.
importSheets {srcProj "/export/home/design1/replace_test/copy1/ori.cpm" lib "worklib"
cell "ori" pages "1-4,7,8,9" replacePages "10-13,16,15,14" type "syscap" destBlockName
"ori" }
```

Related Commands

importBlock

importTechFile

It imports a technology file, which is an ASCII file and is read into a design to specify the user-preferred units, constraint and parameter values, and user properties.

Return Type

NONE

Syntax

```
importTechFile <design_name> <tech_file_path> <overwrite_mode>
```

Parameters

Parameter	Type	Description
`<design_name>`	`STRING`	Name of the design This parameter is required.
`<tech_file_path>`	`STRING`	Path of the technology file This parameter is required.
`<overwrite_mode>`	`BOOL`	Whether to enable overwrite or not. Use 1 to overwrite. This parameter is required.

Examples

```
#command to import the technology file located at D: drive named
#technology_file.tcfx to a design named top:
importTechFile top "D:\techology_file.tcfx" 1
```

importTechnologyFileDialog

It opens the Import Technology File dialog box.

Return Type

NONE

Syntax

```
importTechnologyFileDialog
```

Examples

```
importTechnologyFileDialog
```

insertMultiplePagesAfterCurrentPage

It inserts a specified number of pages after the current page.

Return Type

BOOL

Syntax

```
insertMultiplePagesAfterCurrentPage ?<page_path>? SCH PAGE <num_of_pages>
```

Parameters

Parameter	Type	Description
`page_path`	`STRING`	Path to the current working page. This value must be specified within curly braces {}. This parameter is optional. Default value is `current page`.
`num_of_pages`	`INT`	Number of pages. This value must be specified within curly braces {}. This parameter is required.

Examples

```
insertMultiplePagesAfterCurrentPage {@worklib.doc1(tbl_1):page(1)} SCH PAGE {3}
```

Related Commands

insertMultiplePagesBeforeCurrentPage

insertMultiplePagesBeforeCurrentPage

It inserts a specified number of pages before the current page.

Return Type

BOOL

Syntax

```
insertMultiplePagesBeforeCurrentPage ?<page_path>? SCH PAGE <num_of_pages>
```

Parameters

Parameter	Type	Description
`page_path`	`STRING`	Path of the current working page. This value must be specified within curly braces {}. This parameter is optional.
`num_of_pages`	`INTEGER`	Number of pages. This value must be specified within curly braces {}. This parameter is required.

Examples

```
insertMultiplePagesBeforeCurrentPage {@worklib.doc1(tbl_1):page(2)} SCH PAGE {3}
```

Related Commands

insertMultiplePagesAfterCurrentPage

insertNPages

It inserts multiple pages in the schematic.

Return Type

INT

Syntax

```
insertNPages ?{<page_path>}? SCH PAGE {num_of_page}
```

Parameters

Parameter	Type	Description
`{<page_path>}`	`STRING`	Path of the page after which pages are to be added. This parameter is optional.
`SCH PAGE`	`STRING`	This value is used for a schematic page. This parameter is required.
`{num_of_page}`	`INT`	Number of pages which are to added. This parameter is required.

Examples

```
insertNPages {@worklib.alpha(tbl_1):page(1)} SCH PAGE {6}
```

insertPageAfterCurrentPage

It inserts a page after the current working page.

Return Type

BOOLEAN

Syntax

```
insertPageAfterCurrentPage <page_path> SCH PAGE
```

Parameters

Parameter	Type	Description
`page_path`	`STRING`	Path of the page after which a page is to be added. This parameter is required.
`SCH PAGE`	`STRING`	This value is used for a schematic page. This parameter is required.

Examples

```
insertPageAfterCurrentPage @worklib.doc1(tbl_1):page(2) SCH PAGE
```

Related Commands

insertPageBeforeCurrentPage

insertPageBeforeCurrentPage

It inserts a page before the current working page.

Return Type

BOOLEAN

Syntax

```
insertPageBeforeCurrentPage <page_path> SCH PAGE
```

Parameters

Parameter	Type	Description
`page_path`	`STRING`	Path of the page before which pages are to be added. This parameter is required.
`SCH PAGE`	`STRING`	This value is used for a schematic page. This parameter is required.

Examples

```
insertPageBeforeCurrentPage @worklib.doc1(tbl_1):page(2) SCH PAGE
```

Related Commands

insertPageAfterCurrentPage

invokeBomHDL

Invokes the BOM-HDL application.

Return Type

BOOL

Syntax

```
invokeBomHDL
```

Examples

```
invokeBomHDL
```

invokeVedit

Invokes the Variant Editor application.

Return Type

BOOL

Syntax

```
invokeVedit
```

Examples

```
invokeVedit
```

isActionEnabledInNonElectricMode

This command can be used to check whether provided action is enabled in non-electric state or not.

Return Type

INT

Syntax

```
isActionEnabledInNonElectricMode <context> <actionId>
```

Parameters

Parameter	Type	Description
`context`	`STRING`	Context in which action was created This parameter is required.
`actionId`	`INT`	ID of action This parameter is required.

Examples

```
# Check action4 is enabled in non-electric mode
set customAction4 {Custom Action 4}
set actionId4 [getActionId sch $customAction4]
set ret isActionEnabledInNonElectricMode sch $actionId4

# ret value 1 specifies action is enabled for non-electric state.
# ret value 0 specifies action is disabled for non-electric state.
```

Related Commands

setActionEnabledInNonElectricMode

isActionEnabledInReadOnlyState

Checks whether the provided action is enabled in the read-only state or not.

Return Type

BOOL

Syntax

```
isActionEnabledInReadOnlyState <context> <actionId>
```

Parameters

Parameter	Type	Description
`context`	`STRING`	Context in which action was created This parameter is required.
`actionId`	`INT`	ID of action This parameter is required.

Examples

```
# Check action4 is enabled in read-only mode
set customAction4 {Custom Action 4}
set actionId4 [getActionId sch $customAction4]
set ret isActionEnabledInReadOnlyState sch $actionId4

# ret value 1 specifies action is enabled for read-only state.
# ret value 0 specifies action is disabled for read-only state.
```

Related Commands

setActionEnabledInReadOnlyState

isObjectReadOnly

Checks if the specified object is read-only or not. If it is read-only then returns 1 else 0.

Return Type

BOOL

Syntax

```
sch::isObjectReadOnly <itemId>
```

Parameters

Parameter	Type	Description
`itemId`	`INT`	Item ID of the object that needs to checked if it is read-only This parameter is required.

Examples

```
# This command will return the itemId of the selected block
set blockId [sch::dbGetSPathForActiveTab]
puts $blockId
# Following is an example of itemId of the selected block
# @worklib.ref_5g(tbl_1)

# This command will check if block ref_5g is read-only or not
sch::isObjectReadOnly $blockId
# This will return 1 if block is read-only and 0 if not
```

Related Commands

dbGetSPathForActiveTab

isProjectCached

if yes then return 1
if no then return 0

Return Type

BOOL

Syntax

```
isProjectCached
```

Examples

```
isProjectCached
```

isProjectOpen

Checks if a project is open

Return Type

BOOL

Syntax

```
cps::isProjectOpen
```

Examples

```
puts "Project is open: [cps::isProjectOpen]"
```

isValidPage

Checks whether a page is valid or not. Returns 1, if it is, else NONE.

Return Type

BOOL

Syntax

```
isValidPage <blockName><pageName>
```

Parameters

Parameter	Type	Description
`<blockName>`	`STRING`	block name inside which page lies This parameter is required.
`<pageName>`	`STRING`	page name to be checked This parameter is required.

Examples

```
isValidPage workshop1 page(7)
#Output: 1
```

isValidSDAProject

Checks if the project specified is an Allegro System Capture project

Return Type

BOOL

Syntax

```
cps::isValidSDAProject <path to project>
```

Parameters

Parameter	Type	Description
`path to project`	`STRING`	Path to the project This parameter is required.

Examples

```
cps::isValidProject /home/user/projectpath/proj.cpm
```

lastPage

It opens the last page of the currently active block. 0 is returned on success and an empty list is returned on failure.

Return Type

INT

Syntax

```
sch::lastPage
```

Examples

```
#in the following snippet, the last page of the current block is opened,
#otherwise a message is printed.
if { [sch::lastPage] == {} } { puts "Last page is already open." }
```

Related Commands

firstPage

nextPage

previousPage

launchBoardFile

Opens the board file into the PCB Allegro Editor.

Return Type

NONE

Syntax

```
launchBoardFile <file_name>
```

Parameters

Parameter	Type	Description
`<file_name>`	`STRING`	Full path with name of the Board file This parameter is required.

Examples

```
launchBoardFile "d:/test_1/output/t1/physical/test.brd"
launchBoardFile "../output/t1/physical/test.brd"
```

Related Commands

exportPhysical

launchFindReplace

It launches the Find and Replace dialog box with the Component tab active.

Return Type

NONE

Syntax

```
sch::launchFindReplace
```

Examples

```
#command to open the Find and Replace dialog.
sch::launchFindReplace
```

launchFolderBrowser

It opens the platform directory chooser dialog.

Return Type

STRING

Syntax

```
cps::launchFolderBrowser <title> <initialDirectory>
```

Parameters

Parameter	Type	Description
`title`	`STRING`	Name shown on the file chooser title bar This parameter is required.
`initialDirectory`	`STRING`	Directory to open on launching file chooser dialog This parameter is required.

Examples

```
puts [cps::launchFolderBrowser "TCL FILES" "."]
```

launchUserPreferences

Launches user preference dialog box

Return Type

NONE

Syntax

```
sch::launchUserPreferences
```

Examples

```
sch::launchUserPreferences
```

link

Creates or deletes a link.

Return Type

BOOL

Syntax

```
link ?[-pg <page_name>]? ?[-type <link_item_type>]? <option> [list <x-coordinate> <y-
coordinate>]
```

Parameters

Parameter	Type	Description
`[-pg <page_name>]`	`STRING`	Page name. This parameter is optional. Default value is `Current page`.
`[-type <link_item_type> ]`	`STRING`	Type of the link item. This parameter is optional.
`<option>`	`INT`	The option to add or delete the link. This parameter is required.
`[list <x-coor> <y-coor>]`	`INT`	The XY-coordinates of the position where the link is added. This parameter is required.

Examples

```
link -pg page(15) -type pin -c [list 10316 17389]
```

listAttachments

It shows the list of the all the attachment files present in the currently opened design. It returns the count of the total number of attachment files present in the design and also the attachment file name.

Return Type

STRING

Syntax

```
sch::listAttachments
```

Examples

```
#lists the attachment files present in the currently opened design
#Some attachment files present
sch::listAttachments
#File Listing:
#error.txt
#test.png
#test2.png
#Total Files: 3
#no attachment files present
sch::listAttachments
#Total Files: 0
```

Related Commands

deleteAllAttachments

deleteAttachment

getAllAttachments

getAttachment

storeAttachment

loadSkillFile

It loads the SKILL procedure file indicated by the file path provided. It returns 0 when the SKILL file is loaded successfully and 1 when SKILL file loading is failed. If the file path provided is invalid or inaccessible, the load will fail. If the SKILL file is syntactically incorrect, the loading will fail.

Return Type

INT

Syntax

```
loadSkillFile <file>
```

Parameters

Parameter	Type	Description
`file`	`STRING`	File path of the SKILL procedure file to be loaded This parameter is required.

Examples

```
loadSkillFile {D:\Skill_procedure_location\SkillFile.il}
#Output: 0
```

Related Commands

callSkillFunc

loadURL

It loads the content of URL in the component widget for which handle has been provided.

Return Type

INT

Syntax

```
cps::loadURL <componentHandle> <url>
```

Parameters

Parameter	Type	Description
`componentHandle`	`INT`	Handle of component to host URL This parameter is required.
`url`	`STRING`	URL to be loaded This parameter is required.

Examples

```
#command to load contents of yourblog.com in hybrid tab myblog.
set tabHandle [ sdaUI::hybridTab {myblog} {http://myblog.com} {blogid} {blogname} ]
cps::loadURL $tabHandle {http://yourblog.com}
```

Related Commands

createHybrid

makeBlockReadOnly

Makes a writable block read-only.

Return Type

BOOLEAN

Syntax

```
makeBlockReadOnly <block_name>
```

Parameters

Parameter	Type	Description
`<block_name>`	`STRING`	name of the block to be made read-only This parameter is required.

Examples

```
makeBlockReadOnly block_name
#Output: 1
```

makeBlockViewWritable

Unlocks the read-only block's symbol view.

Return Type

BOOLEAN

Syntax

```
makeBlockViewWritable <block_name> <view_name>
```

Parameters

Parameter	Type	Description
`block_name`	`string`	Block name of the symbol to be unlocked for editing This parameter is required.
`view_name`	`string`	Symbol view to be unlocked for editing This parameter is required.

Examples

```
makeBlockViewWritable clock sym_1
```

makeBlockWritable

Makes a read-only block writable.

Return Type

BOOLEAN

Syntax

```
makeBlockWritable <block_name>
```

Parameters

Parameter	Type	Description
`<block_name>`	`STRING`	Name of the block which needs to be made writable This parameter is required.

Examples

```
makeBlockWritable block_name
```

mirror

Creates a mirror image of the selected element(s) on the current page.

Return Type

BOOLEAN

Syntax

```
mirror <type>
```

Parameters

Parameter	Type	Description
`<type>`	`STRING`	type of mirror alignment This parameter is required.

Examples

```
mirror horizontal
```

modifyCustomVar

Modifies the value of the custom variable.

Return Type

NONE

Syntax

```
modifyCustomVar -name <variable_name> -value <variable_new_value> -type <value>
```

Parameters

Parameter	Type	Description
`-name <variable_name>`	`STRING`	Variable name. Variable name has to be in uppercase. This parameter is required.
`-value <variable_new_value>`	`STRING`	New value of the variable. Enclose within double quotes (" "). This parameter is required.
`-type <value>`	`STRING`	Type of custom variable, Tool Defined or Project Defined. Enclose within double quotes (" "). This parameter is required.

Examples

```
modifyCustomVar -name ALPHA -value "beta" -type "Project Defined"
```

modifyProp

Changes the properties of the specified components. Can be used to change the visibility of multiple properties at once. For this, provide a list of property names in the -name argument.

Return Type

NONE

Syntax

```
modifyProp -name {value} -value {value} ?-scope {value}? ?-scope2 {value}? ?-display
"value" ? ?-mode 1?
```

Parameters

Parameter	Type	Description
`-name {value}`	`STRING`	The name of the property. This parameter is required.
`-value {value}`	`STRING`	The value of the property. This parameter is required.
`-scope {value}`	`STRING`	The scope of the search for the specified object, the entire design or the current block. This parameter is optional.
`-scope2 {value}`	`STRING`	Were to search for the specified object - whether to search for in components or nets This parameter is optional.
`-mode`	`INT`	Type of the match which is wildcard This parameter is optional.
`-regex`	`NONE`	Use regular expression for matching This parameter is optional.
`-strict`	`NONE`	Type of the match is exact matching This parameter is optional.

Examples

```
modifyProp -name PACK_TYPE -value FBGA97
modifyProp -name [list {VER} {XY}] -display valOnly
modifyProp -name [list {VER} {XY}] -display nameOnly
#batch visibility change command
#options for scope: entire design or block or selection
#options for scope2: Component or Nets
#options for type of matching: -mode 1[wild card] or -regrex [regular expression] or -
strict [excat match]
modifyProp -scope "Entire Design" -scope2 "Component" -name "PACK_TYPE " -value "*" -
display "nodisp" -mode 1
# match type is regular expression
modifyProp -scope "Entire Design" -scope2 "Component" -name "PACK_TYPE " -value "^val"
-display "nodisp" -regrex
# match type is eaxct match
modifyProp -scope "Entire Design" -scope2 "Component" -name "PACK_TYPE " -value "value"
-display "nodisp" -strict
```

MTBFPreferences

Opens the Design MTBF Settings dialog box.

Return Type

NONE

Syntax

```
sdaReliability::MTBFPreferences
```

Examples

```
sdaReliability::MTBFPreferences
```

Related Commands

runMTBFAnalysis

newProject

Creates a new project. Returns 0 if the command is successful. The newly created project gets automatically opened.

Return Type

BOOL

Syntax

```
newProject <project_name> <design_name> <project_path> sch composite
```

Parameters

Parameter	Type	Description
`project_name`	`STRING`	Name of the project to be created. The project CPM file gets created with this name This parameter is required.
`design_name`	`STRING`	Name of the root design. This parameter is required.
`project_path`	`STRING`	Specify the absolute path to the location where the project is to be created. This parameter is required.
`sch`	`STRING`	Indicates that the schematic view will be created. No other value is supported for this parameter. This parameter is required.
`composite`	`STRING`	Mandatory for creating a standard System Capture project. No other value is supported for this parameter. This parameter is required.

Examples

```
#create a System Capture project named 'myproject', that has 'root' as
#the design name. The project should be saved in the 'projects' folder.
#Linux example
newProject "myproject root {/home/user/projects} sch composite"
#Windows example
newProject "myproject root {d:/projects} sch composite"
```

nextPage

It will open the next page in the schematic and will work with the page numbers. If the current page is page 3, then it will open the page 4 (If it exists).

Return Type

STRING

Syntax

```
sch::nextPage
```

Examples

```
#this command will open the next page on the schematic.
#If you are on the last page already then it will return
#empty string else will return 0.
sch::nextPage
```

Related Commands

previousPage

openItem

Opens the specified object whose itemId is passed in a new tab.

The following values are supported for viewType ItemType pairs:
SCH PAGE - To open a schematic page
FILE TEXT - To open a text file
START_PAGE START_PAGE - To open a Start Page for System Capture

Return Type

NONE

Syntax

```
openItem <itemId> <viewType > <itemType> ?activateTab?
```

Parameters

Parameter	Type	Description
`itemId`	`STRING`	Unique identifier for the object to be opened This parameter is required.
`viewType`	`STRING`	Predefined string identifying the type of object to be opened. SCH and FILE are supported view types. This parameter is required.
`itemType`	`STRING`	Predefined string identifying the subtype of object to be opened. PAGE and TEXT are supported item types. This parameter is required.
`activateTab`	`BOOL`	Specifies whether the newly opened tab should become the active tab. This parameter is optional. Default value is `1`.

Examples

```
# This command will return the itemId of the current active item
sch::dbGetActivePageSpath
# If ref_5g sample project is opened, this command will return the itemId of the
current active item, in example shown below.
# @worklib.ref_5g(tbl_1):Audio(3)

# This command will open the schematic page "Audio(3)" of block "ref_5g"
openItem {@worklib.ref_5g(tbl_1):Audio(3)} SCH PAGE
```

Related Commands

dbGetActivePageSpath

dbGetSPathForActiveTab

openPrintUI

Use openPrintUI to open the print menu.

Return Type

NONE

Syntax

```
sch::openPrintUI
```

Examples

```
sch::openPrintUI
```

openProject

Opens an existing project.

Return Type

NONE

Syntax

```
openProject <project_path> <open_default>
```

Parameters

Parameter	Type	Description
`<project_path>`	`STRING`	Project path enclosed within straight double quotes (" "). This parameter is required.
`<open_default>`	`INTEGER`	Default value. Always 1. This parameter is required.

Examples

```
openProject "D:/ASDA_CPMX/newproj.cpm" 1
```

openURLDialog

Use openURLDialog to display a web page within a dialog.

Return Type

INT

Syntax

```
sdaUI::openURLDialog <title> <url> <name> ?<icon>? ?<position>? ?<size>? ?<isModal>? ?
<isResizable>?
```

Parameters

Parameter	Type	Description
`title`	`STRING`	Title of the tab This parameter is required.
`url`	`STRING`	URL of the content to host This parameter is required.
`name`	`STRING`	Name of the hybrid component. This name can be used for looking up the component handle using cps::findComponentByName This parameter is required.
`icon`	`STRING`	Path to the dialog icon This parameter is optional. Default value is `{}`.
`position`	`LIST`	Position of the web-view within the dialog. This is reserved for future use This parameter is optional. Default value is `{0 0}`.
`size`	`LIST`	Size of the dialog in width and height This parameter is optional. Default value is `{800 800}`.
`isModal`	`BOOL`	Modality of the dialog. A 1 would indicate a modal dialog This parameter is optional. Default value is `1`.
`isResizable`	`BOOL`	Resizability of the dialog This parameter is optional.

Examples

```
# This command will create a dialog with the below URL hosting in it.
sdaUI::openURLDialog CADENCE www.cadence.com testtab02 {} {0 0} {1000 400} 1 1
# It will return the dialog handle of the dialog created.
# Output: 78
```

Related Commands

openURLDialogWithCloseHandler

openURLTab

It displays a web page within a tabbed view. On success, returns a non-negative integer representing the unique ID of the newly created tab. This ID can be subsequently used to execute JavaScript functions on that tab using the cps::execute command.

Return Type

INT

Syntax

```
sdaUI::openURLTab <title> <url> <id> <name>
```

Parameters

Parameter	Type	Description
`title`	`STRING`	Title of the tab This parameter is required.
`url`	`STRING`	URL of the content to host This parameter is required.
`id`	`STRING`	Unique identifier for the tab This parameter is required.
`name`	`STRING`	Name of the hybrid component. This name can be used for looking up the component handle using cps::findComponentByName This parameter is required.

Examples

```
set widgetId [sdaUI::openURLTab customWidget "file:///home/johndoe/atest.html" testid01
MyTestId]
cps::execute $widgetId "sayhello()"
#where sayhello is a JavaScript function defined within the atest.html file.
```

pageSetup

It changes the page size of the current page.

Return Type

NONE

Syntax

```
pageSetup -pg <page_name> <type> -standard <page standard> -zones<zone enabled>
<horizontal count> <vertical count> <label ordering>; [zoom -fit]
```

Parameters

Parameter	Type	Description
`-pg<page_name>`	`STRING`	Name of the current page This parameter is required.
`<type>`	`STRING`	One of the predefined page sizes: A to E This parameter is required.
`<page_standard>`	`STRING`	page standard name to which the page type belongs This parameter is required.
`<zone enabled>`	`BOOL`	enable disable zones This parameter is required.
`<horizontal_count>`	`INT`	horizontal zones count This parameter is required.
`<vertical count>`	`INT`	vertical zones count This parameter is required.
`<label ordering>`	`STRING`	horizontal and vertical zones label order This parameter is required.

Examples

```
pageSetup @worklib.workshop1(tbl_1):page(2) A
pageSetup -pg page(2) custom 42.2 32.2;zoom -fit
pageSetup -pg page(2) symbol "standard" "a#20size#20page" "sym_1";zoom -fit
pageSetup -pg page(1) B -standard ANSI
#the above example creates a page size B defined in ANSI standard.
#-standard can only take standard defined by cpm directive 'PAGE_DEFAULT_STANDARD' as
parameter.
#-standard option must always be passed to create a page from new standards introduced
in 17.41.
#below is the list of supported standards and their respective page sizes.
#i) ANSI
#supported page sizes A,B,C,D,E
#ii) ISO
#supported page sizes A0,A1,A2,A3,A4
#iii) JIS
#supported page sizes B0,B1,B2,B3,B4
#iV) GOST
#supported page sizes A0,A1,A2,A3,A4
pageSetup -pg page(1) -zones true 8 8 BTLLRN
#the above example enables zones on page(1) with horizontal and vertical row count 8,8
#and with zone labeling order BTLLRN.
#below is the full-form of zone labeling order
#Top to Bottom Letters, Left to Right Numbers : TBLLRN
#Bottom to Top Letters, Left to Right Numbers : BTLLRN
#Top to Bottom Letters, Right to Left Numbers : TBLRLN
#Bottom to Top Letters, Right to Left Numbers : BTLRLN
#Top to Bottom Numbers , Left to Right Letters : TBNLRL
#Bottom to Top Numbers , Left to Right Letters : BTNLRL
#Top to Bottom Numbers , Right to Left Letters : TBNRLL
#Bottom to Top Numbers , Right to Left Letters : BTNRLL
```

paste

Pastes the contents of the clipboard at the specified or current cursor position.

Return Type

NONE

Syntax

```
paste -index <value> -pg <page_name> [list <x-coordinate> <y-coordinate>]
```

Parameters

Parameter	Type	Description
`-index <value>`	`INT`	SDA defined value. By default 1. This parameter is required.
`-pg <page_name>`	`STRING`	The name of the page where the object(s) is to be pasted. This parameter is required.
`[list <x-coordinate> <y-coordinate>]`	`INT`	The XY-coordinates of the position where the object is to be pasted. This parameter is required.

Examples

```
paste -index 1 -pg @worklib.workshop1(tbl_1):page(6) [list 3550 3460] -op [list ]
```

pasteAfterCurrentPage

Paste the copied page after the current page.

Return Type

BOOL

Syntax

```
pasteAfterCurrentPage "<currentPagePath>"
```

Parameters

Parameter	Type	Description
`currentPagePath`	`STRING`	Path of the page after which a page will be pasted This parameter is required.

Examples

```
pasteAfterCurrentPage @worklib.mid_test(tbl_1):page(2)
```

pasteBeforeCurrentPage

Paste the page copied before the current page.

Return Type

BOOL

Syntax

```
pasteBeforeCurrentPage "<currentPagePath>"
```

Parameters

Parameter	Type	Description
`currentPagePath`	`STRING`	Path of current page before which a page will be pasted This parameter is required.

Examples

```
pasteBeforeCurrentPage @worklib.mid_test(tbl_1):page(2)
```

pptOptions

Changes the physical part table.

Return Type

BOOLEAN

Syntax

```
pptOptions [-match_case_sensitive-ptf <choice>] [-use_library_ppt <choice>] [-merge_ppt
<choice>]
```

Parameters

Parameter	Type	Description
`[-match_case_sensitive-ptf <choice>]`	`STRING`	To perform case sensitive row match. This parameter is optional.
`[-use_library_ppt <choice>]`	`STRING`	To use cell level physical part table files. This parameter is optional.
`[-merge_ppt <choice>]`	`STRING`	To merge physical part table files. This parameter is optional.

preferenceSpecialBodies

Add a symbol to the special symbols palette. Library, cell, view should be specified in the format defined in the syntax. The library, cell, view syntax for power/ground symbol is library:cell:view, whereas for the other types, it is library.cell:view.

For power/ground symbols, cell names starting with "gnd" or "ground" are added to the Ground section, otherwise to the Power section of the special symbols palette.

Return Type

NONE

Syntax

```
preferenceSpecialBodies [-inportSymbol <library.cell:view>] [-outportSymbol
<library.cell:view>] [-ioportSymbol <library.cell:view>] [-offpageInputSymbol
<library.cell:view>] [-offpageOutputSymbol <library.cell:view>] [-offpageIOSymbol
<library.cell:view>] [-defaultTapSymbol <library.cell:view>] [-ninetyDegreeTapSymbol
<library.cell:view>] [-oneEightyDegreeTapSymbol <library.cell:view>] [-
twoSeventyDegreeTapSymbol <library.cell:view>] [-aliasBodySymbol <library.cell:view>]
[-commentBodySymbol <library.cell:view>] [-noConnectBodySymbol <library.cell:view>] [-
powerGroundSymbol <library:cell:view>]
```

Parameters

Parameter	Type	Description
`inportSymbol`	`STRING`	Library, cell, view of the input port symbol This parameter is required.
`outportSymbol`	`STRING`	Library, cell, view of the output port symbol This parameter is required.
`ioportSymbol`	`STRING`	Library, cell, view of the I/O port symbol This parameter is required.

`offpageInputSymbol`	`STRING`	Library, cell, view of the input off-page symbol This parameter is required.
`offpageOutputSymbol`	`STRING`	Library, cell, view of the output off-page symbol This parameter is required.
`offpageIOSymbol`	`STRING`	Library, cell, view of the I/O off-page symbol This parameter is required.
`defaultTapSymbol`	`STRING`	Library, cell, view of the default bus-tap symbol This parameter is required.
`ninetyDegreeTapSymbol`	`STRING`	Library, cell, view of the 90 degree oriented bus-tap symbol This parameter is required.
`oneEightyDegreeTapSymbol`	`STRING`	Library, cell, view of the 180 degree oriented bus-tap symbol This parameter is required.
`twoSeventyDegreeTapSymbol`	`STRING`	Library, cell, view of the 270 degree oriented bus-tap symbol This parameter is required.
`aliasBodySymbol`	`STRING`	Library, cell, view of the alias/synonym symbol This parameter is required.
`commentBodySymbol`	`STRING`	Library, cell, view of comment body symbol This parameter is required.
`noConnectBodySymbol`	`STRING`	Library, cell, view of no-connect symbol This parameter is required.
`powerGroundSymbol`	`STRING`	Library, cell, view of the power or ground symbol This parameter is required.

Examples

```
# add a single symbol of a specific type to the palette
preferenceSpecialBodies -inportSymbol standard.outport:sym_1
preferenceSpecialBodies -offpageInputSymbol standard.offpage:sym_1
preferenceSpecialBodies -oneEightyDegreeTapSymbol standard.tap:sym_1
preferenceSpecialBodies -aliasBodySymbol standard.alias:sym_1
preferenceSpecialBodies -powerGroundSymbol standard:vcc_circle:sym_1

# add multiple symbols to the special symbols palette
preferenceSpecialBodies -inportSymbol standard.outport:sym_1 -outportSymbol
standard.inport:sym_1 -ioportSymbol standard.ioport:sym_1 -offpageInputSymbol
standard.offpage:sym_1 -offpageOutputSymbol standard.offpage:sym_2 -offpageIOSymbol
standard.offpage:sym_3 -defaultTapSymbol standard.tap:sym_6 -ninetyDegreeTapSymbol
standard.tap:sym_7 -oneEightyDegreeTapSymbol standard.tap:sym_1 -
twoSeventyDegreeTapSymbol standard.tap:sym_3
```

preview

it will create instance on scene

Return Type

BOOL

Syntax

```
preview <object_name>
```

previousPage

It will open the previous page in the schematic and will work with the page numbers. If current page is page 3, then it will open page 2, irrespective of the page that was last opened.

Return Type

STRING

Syntax

```
sch::previousPage
```

Examples

```
#command to open the previous page on the schematic.
#If you are on the first page already then it will return
#empty string else will return 0.
sch::previousPage
```

Related Commands

nextPage

print

Prints the schematic design. You can take printouts, generate a basic PDF, Smart PDF, or a PDF/A.

Return Type

NONE

Syntax

```
print -printer <printer_name> -noofCopies <int_value> -orientation <orientation_string>
-pageSize <page_size> -fitToPage <bool_value> -range <page_range> -colorMode
<int_value> -printToPdf <boolean_string> -printToSmartPdf <boolean_string> -
scaleContents <boolean_string> -scaleFactor <scaling_factor> -pdfFilePath
<pdf_file_path> -encrPwd <Pdf_password> -printAllowed <int_value> -changeAllowed
<int_value> -enableCopy <boolean_string> -enableSoundRead <boolean_string> -margin
<page_margin_list> -watermarkText <watermark_text> -watermarkRotation <int_value> -
watermarkOpacity <int_value> -disableSmartFeatures <boolean_string> -enablePDFAFeature
<boolean_string>
```

Parameters

Parameter	Type	Description
`printer`	`STRING`	Name of the printer as seen in the Operating System printer list This parameter is required.
`noofCopies`	`INT`	Number of copies to be printed. This parameter is required.
`orientation`	`STRING`	Defines the orientation of the page which is to be printed. Supported values are Landscape or portrait. This parameter is required.

`pageSize`	`STRING`	Defines the size of the page, this value depends on the type of sizes available on canvas. This parameter is required.
`fitToPage`	`BOOL`	Defines either the PDF pages will be fit to page or not. This parameter is required.
`range`	`LIST`	Defines the range of pages to be printed from the entire design. For example, if values are [1 5] then PDF will be generated from page 1 to page 5. This parameter is required.
`colorMode`	`INT`	This property tells the color visibility of the contents of PDF. Supported values are 0, 1, 2. 0 means color same as that on canvas, 1 means content will be in gray color and 2 means the PDF will be in black and white. This parameter is required.
`printToPdf`	`BOOL`	This property defines if the pdf generated will be normal PDF or not. If true then normal PDF else SmartPDF. Parameter "printToPdf" and "printToSmartPdf" are opposite to each other. This parameter is required.
`printToSmartPdf`	`BOOL`	This property defines if the pdf generated will be SmartPDF or not. If true then SmartPDF will be generated else normal PDF. Parameter "printToPdf" and "printToSmartPdf" are opposite to each other.. This parameter is required.
`scaleContents`	`BOOL`	This property defines if the pdf generated will be scaled or not before printing. This parameter is required.
`scaleFactor`	`INT`	This property is dependent on <scaleContents> parameter, if scaling is allowed then this parameter defines by how much scaling needs to be done. This parameter is required.

`pdfFilePath`	`STRING`	The path where PDF file will be created. This parameter is required.
`encrPwd`	`STRING`	This argument displays the password set to open the required PDF generated. This feature will be disabled when smart features are disabled. This feature is not supported with PDF/A is generated. This parameter is required.
`printAllowed`	`INT`	This parameter is for security purposes only, this is available only when encryption is enabled while generating PDF. Supported values are None, High Resolution and Low Resolution. If set "None" printing of generated PDF will not be allowed, if set "High Resolution" then PDF will be generated in high resolution state and if set as "Low Resolution" PDF will be generated in low resolution state. This parameter is required.
`changeAllowed`	`INT`	This parameter is for security purposes, this is available only when encryption is enabled while generating PDF. This parameter defines if externally changes can be done while printing the PDF. This parameter is required.
`enableCopy`	`BOOL`	This parameter when will allow user to copy the contents of PDF generated. This parameter is available when encryption is enabled while generating PDF. This parameter is optional. Default value is `false`.
`enableSoundRead`	`BOOL`	This parameter when will restrict enable PDF reading option in any PDF viewer(say Adobe). This parameter only when encryption is enabled while generating PDF. This parameter is optional. Default value is `false`.

`margin`	`LIST`	This property sets the top, bottom, left and right margins around the border while generating PDFs. This parameter is required.
`watermarkText`	`STRING`	This property displays the watermark text which will be visible while generating PDF, this feature is not available for PDF/A. This parameter is required.
`watermarkRotation`	`INT`	This property defines the orientation of the watermark text. Supported values are 0 degree and 45 degree. This parameter is optional.
`watermarkOpacity`	`INT`	This property defines the transparency level of watermark text. Range of values are 0 - 100, moving from 100 to 0 decrease the visibility of watermark text. This parameter is required.
`disableSmartFeatures`	`BOOL`	This feature defines if the PDF generated will contain smart features or not. If set true then PDF will not contain smart features. This parameter is required.
`enablePDFAFeature`	`BOOL`	This feature defines the generated PDF will be PDF/A complaint or not. If set true then it is PDF/A else SmartPDF. This parameter is required.

Examples

```
# This command to generates the SmartPDF with smart features.
# "disableSmartFeatures" parameter is set as false for SmartPDF generated with smart
features.
print -printer {Smart PDF} -noofCopies 1 -orientation Landscape -pageSize {Same as
source} -fitToPage true -range [list 1 1] -colorMode 1 -printToPdf false -
printToSmartPdf true -scaleContents false -scaleFactor 100 -pdfFilePath
{D:/users/mraduls/CCRs/1PDFA_CCRs/syntax_error_real_value_outofrange/attolon_mb_v1/1.pd
f} -margin [list 0.5 0.5 0.5 0.5] -watermarkText {} -watermarkRotation 1 -
watermarkOpacity 10 -blockNames [list attolon_mb_v1] -disableSmartFeatures false -
enablePDFAFeature false

# This command generates the SmartPDF without smart features.
# "disableSmartFeatures" parameter is set as true for SmartPDF generated without smart
features.
print -printer {Smart PDF} -noofCopies 1 -orientation Landscape -pageSize {Same as
source} -fitToPage true -range [list 1 1] -colorMode 1 -printToPdf false -
printToSmartPdf true -scaleContents false -scaleFactor 100 -pdfFilePath
{D:/users/mraduls/CCRs/1PDFA_CCRs/syntax_error_real_value_outofrange/attolon_mb_v1/1.pd
f} -margin [list 0.5 0.5 0.5 0.5] -watermarkText {} -watermarkRotation 1 -
watermarkOpacity 10 -blockNames [list attolon_mb_v1] -disableSmartFeatures true -
enablePDFAFeature false

# This command generates the PDF/A, "enablePDFAFeature" is set as true for PDF/A
# PDF/A by default has smart features enabled
print -printer {Smart PDF} -noofCopies 1 -orientation Landscape -pageSize {Same as
source} -fitToPage true -range [list 1 1] -colorMode 1 -printToPdf false -
printToSmartPdf true -scaleContents false -scaleFactor 100 -pdfFilePath
{D:/users/mraduls/CCRs/1PDFA_CCRs/syntax_error_real_value_outofrange/attolon_mb_v1/1.pd
f} -margin [list 0.5 0.5 0.5 0.5] -watermarkText {} -watermarkRotation 1 -
watermarkOpacity 10 -blockNames [list attolon_mb_v1] -disableSmartFeatures false -
enablePDFAFeature true
```

printvariant

prints the variant

Return Type

NONE

Syntax

```
printvariant
```

Examples

```
printvariant JAPAN
```

PTreePreferences

Opens the 'Power Topology Settings' dialog box.

Return Type

NONE

Syntax

```
sdaReliability::PTreePreferences
```

Examples

```
sdaReliability::PTreePreferences
```

Related Commands

sdaReliability::extractPTree

queryPartPropsFromSpath

Gets the part properties for instances with the specified spaths. This command is used after the 'cpb::generatePartPropertiesData' has been completed successfully. It returns a list of property name and value pairs.

Return Type

list

Syntax

```
cpb::queryPartPropsFromSpath spath
```

Parameters

Parameter	Type	Description
`spath`	`STRING`	spath of item for which part properties are to be fetched This parameter is required.

Examples

```
#Get a list of spaths of items selected on canvas and query part properties
set spath [ sch::dbGetSPath [ sch::dbGetSelectedItems [sch::dbGetActivePage ] ] ] ]
set vecList [cps::stdVectorStr]
foreach item $spath { $vecList push $item }
cpb::generatePartPropertiesData 1 $vecList
foreach item $spath {
set propList [ cpb::queryPartPropsFromSpath $item ]
}
```

Related Commands

generatePartPropertiesData

redo

Reverses the effects of the previous undo command. You can reverse multiple actions that have been undone. A redo command can be used only after the undo command.

Return Type

BOOLEAN

Syntax

```
redo
```

Examples

```
redo
```

Related Commands

undo

reevaluateTOC

Updates the table of contents page. You need to select at least one of the tables on the TOC page.

Return Type

BOOLEAN

Syntax

```
reevaluateTOC
```

Examples

```
reevaluateTOC
```

refreshSymbol

It refreshes the symbol of the selected component from the library if library, cell, view are blank otherwise it works for given data

Return Type

BOOLEAN

Syntax

```
refreshSymbol
```

Parameters

Parameter	Type	Description
`library`	`STRING`	library of a part, optional parameter with default value is BLANK This parameter is optional.
`cell`	`STRING`	cell of a part, optional parameter with default value is BLANK This parameter is optional.
`view`	`STRING`	view of a part, optional parameter with default value is BLANK This parameter is optional.

Examples

```
1) refreshSymbol =&gt; it works for selected part

2) refreshSymbol &lt;library&gt; &lt;cell&gt; &lt;view&gt; =&gt; it works for given
library data
```

regeneratePhysNetNames

It regenerates the physical net names of the nets in the entire design. It works on the current root design. It is not applicable for blocks that are not instantiated in the current root design.

Return Type

NONE

Syntax

```
regeneratePhysNetNames
```

Examples

```
#command to regenerate physical net names
#for all nets in the design
regeneratePhysNetNames
```

Related Commands

reassignRefdes

exportPhysical

registerCommand

Registers two TCL procedures one as a pre-procedure (preTriggerTclProcedure) and another as a post-procedure (postTriggerTclProcedure) to any System Capture built-in Tcl command.

A built-in command is not a TCL procedure. If the following command is given:
info body <command>
A message states that <command> is not a procedure

Both pre and post Trigger TCL procedures accept the "args" paramater, that is:
proc <nameOfProc> {args} {
<body of proc>
}

If two preTriggerTclProcedure and postTriggerTclProcedure commands are defined, they get called before and after the built-in command, respectively.

Return Type

STRING

Syntax

```
cps::registerCommand <SystemCaptureBuiltInCommand> <preTriggerTclProcedure>
<postTriggerTclProcedure>
```

Parameters

Parameter	Type	Description
`SystemCaptureBuiltInCommand`	`STRING`	Any built-in Tcl command in Allegro System Capture This parameter is required.
`preTriggerTclProcedure`	`STRING`	The TCL procedure to be called just before SystemCaptureBuiltInCommand This parameter is optional.
`postTriggerTclProcedure`	`STRING`	The TCL procedure to be called immediately after SystemCaptureBuiltInCommand This parameter is optional.

Examples

```
# Example - 1
# Defining preTriggerTclProcedure which will be registered with
SystemCaptureBuiltInCommand.
proc preSave {args}
{
puts "pre save"
}

# Defining postTriggerTclProcedure which will be registered with
SystemCaptureBuiltInCommand.
proc postSave {args}
{
puts "post save"
}

# Registering above two procedures with cps::saveAll command. cps::saveAll is System
Capture's built-in TCL command.
cps::registerCommand cps::saveAll preSave postSave

# Now when saveAll TCl command is called in System Capture then preSave procedure will
be called first then saveAll command will be called and just
# after this, postSave procedure is called.
```

```
saveAll
# Following is the output.
# pre
# post
# 0



# Example - 2
# In this example a pre-procedure is attached to the built-in paste command which
controls the paste operation. Here paste operation will be skipped
# when the procedure named prePasteHandler is attached to paste operation. Also, notice
it has no post-procedure as pre-procedure will skip
# execution of paste command so the post-procedure will not be called even if defined.
proc isPasteAllowed {args} {
return 0
}

proc prePasteHandler {args} {
set ret 0
set clipboardData {}
if { [isPasteAllowed $clipboardData] == 0 } {
# if paste is not allowed return SKIP ALL
puts " WARNING - PASTE DISABLED"
set ret $::cps::eCPS_SKIP_ALL
}
return $ret
}

cps::registerCommand paste prePasteHandler { }
# The prePasteHandler stops the paste operation based on business logic by returning
SKIP ALL
```

registerDesignRule

Command to register a new DRC rule in the rule list. On successful run, the rule will be listed in the DRC rule list in project preferences.

Return Type

NONE

Syntax

```
sch::registerDesignRule <ruleName> <ruleDescription> <ruleProc> <ruleType>
```

Parameters

Parameter	Type	Description
`ruleName`	`STRING`	Name of the rule This parameter is required.
`ruleDescription`	`STRING`	Rule description This parameter is required.
`ruleProc`	`STRING`	Function name for the rule This parameter is required.
`ruleType`	`STRING`	Type of rule - logical/electrical This parameter is required.

Examples

```
To register new graphical rule with name DRC_NewRule -
sch::registerDesignRule DRC_NewRule "new rule" drc_new_rule Graphical
```

reImport

It re-imports the block from the source System Capture design.

Return Type

NONE

Syntax

```
reImport srcProj <source_proj_path> cell <cell_name> ?type <source_design_type>? ?-
preserveModifiedSymbols?
```

Parameters

Parameter	Type	Description
`srcProj`	`STRING`	The path to the project file (.cpm or .sdax) from which the block is to be re-imported. This parameter is required.
`cell`	`STRING`	The cell or block name to be re-imported. This parameter is required.
`type`	`STRING`	The source project type. Valid values are 'syscap'. This parameter is required.
`-preserveModifiedSymbols`	`NONE`	Specify whether to preserve the 'Unlocked' symbols of the block. If this option is specified the symbol will not be imported from the source design. If not specified, the symbol will be overwritten. This parameter is optional.

Examples

```
#command to import the 'n2' from a System Capture project.
reImport {srcProj {D:/designs/n2/logic/n2.sdax} cell n2 type syscap }
#command to import the 'n2' block in preserve symbol mode from a
#System Capture project.
reImport {srcProj {D:/designs/n2/n2.cpm} cell n2 type syscap -preserveModifiedSymbols}
```

Related Commands

importBlock

reImportBlock

It opens the import block UI for the specified block.

Return Type

NONE

Syntax

```
reImportBlock <block_name>
```

Examples

```
reImportBlock top
```

Related Commands

reImport

relayoutNavlinks

It works on the selected Port/Offpage. It will re-format all the associated navigation links

Return Type

NONE

Syntax

```
sch::relayoutNavlinks
```

Examples

```
sch::relayoutNavlinks
```

removeAlternatePart

Removes an alternate part of a component for the current variant.

Return Type

NONE

Syntax

```
removeAlternatePart -variant <variant_name> -comp <component_name> -key <properties>
```

Parameters

Parameter	Type	Description
`-variant <variant_name>`	`STRING`	Name of the variant design. This parameter is required.
`-comp <component_name>`	`STRING`	Name of the component. This parameter is required.
`-key <properties>`	`STRING`	Key properties of the alternate part. Each property is separated by a comma(,). This parameter is required.

Examples

```
removeAlternatePart -variant QWERTY -comp C2 -key C0G-CERM,0402-1,CAP,+/-
0.1PF,6.0PF,50V
```

renameSignal

This command renames the wire whose position is specified.

Return Type

BOOL

Syntax

```
renameSignal -pg <pageSPath> -net <netPositionAndName>
```

Parameters

Parameter	Type	Description
`pageSPath`	`STRING`	sPath of the page where the wire is placed This parameter is required.
`netPositionAndName`	`LIST`	List of X,Y coordinates of segment, old name, and new name This parameter is required.

Examples

```
# This command renames the wire drawn at below list of coordinates from oldNet to
newNet
renameSignal -pg @worklib.test_design(tbl_1):page(1) -net [list 3050 13700 {oldNet}
{newNet}]
```

replace

Use this command to replace a component with another component, pin shape, or symbol outline. When replacing pin shape or symbol outline, use the 'selectObject' command to select the pin or symbol outline to be replaced.

Return Type

NONE

Syntax

```
replace -part -by <lib_name>.<cell_name>.<view_name> -key [list <properties>]
preserveRefDes -preserveUserProp | -pinshape -pintype <shape_name> | -outline
<outline_name>
```

Parameters

Parameter	Type	Description
`-part -by`	`STRING`	System-defined This parameter is required.
`<lib_name>.<cell_name>:<view_name>`	`STRING`	The lib.cell:view of the component. This parameter is required.
`-key [list <properties>]`	`STRING`	List of properties. This parameter is required.
`-preserveRefDes -preserveUserProp`	`STRING`	These are System-defined keywords. This parameter is required.

`-retainVoltage`	`STRING`	Only applicable in case of special symbol replace. When the source symbol is a power symbol, this option can be used to retain the existing voltage property of the symbol on the replaced symbol. This parameter is required.
`-pinshape -pintype <shape_name>`	`KEYWORD`	Parameter to indicate that pin shape is to be modified. This parameter is required.
`-pintype <shape_name>`	`STRING`	The shape name which needs to be added. The valid values are: VCCTriangle, PortGroupLine, OutArrowLine2, OutArrowLine1, OutArrow, Line2, Line1, Line, InOutArrowLine2, InOutArrowLine1, InOutArrow, InArrowLine2, InArrowLine1, InArrow, GNDEarth, CrossLine2, CrossLine1, Cross, CircleOutArrowLine, CircleOutArrow, CircleLineSmall, CircleLine2, CircleLine1, CircleLine, CircleInOutArrow, CircleInOutArrowBusR, CircleInArrowLine, CircleInArrow, Circle, Line. This parameter is required.
`-outline <outline_name>`	`STRING`	The outline shape name which replaces the old one. Valid outline shape names are: MUX, Octagon, Rectangle, RightTriangle, Triangle, SchottkyDiaode, ZenerDiaode, Resistor, 3SpikeResistor, Capacitor Vertical, Capacitor Horizontal, Circle, Transistor, NPN Transistor, PNP Transistor. This parameter is required.

Examples

```
replace -part -by vlsi.21555:sym_1 -key [ list ] -preserveRefDes -preserveUserProp
```

```
replace -pinshape -pintype InOutArrow
```

```
replace -outline RectOutline
```

Related Commands

selectObject

replaceComponent

It replaces the components.

This command is no longer in use.

Return Type

BOOL

Syntax

```
replaceComponent
```

Examples

```
replaceComponent ["comp_1" "comp_2"]
```

resetAuditDirective

Use this command to reset the current selection of audit rules to default list of audit rules to run.

Return Type

NONE

Syntax

```
sdaReliability::resetAuditDirective
```

Examples

```
sdaReliability::resetAuditDirective
```

resetImage

It resets the selected image to its original size.

Return Type

NONE

Syntax

```
resetImage
```

Examples

```
resetImage
```

resetLicenseCache

This command resets the license.

Return Type

NONE

Syntax

```
cps::resetLicenseCache
```

route

Converts the selected rat to route segments.

Return Type

BOOLEAN

Syntax

```
route <type>
```

runDBDoctor

Runs the DB Doctor utility in one of the following two modes.

Detect Mode
The detect mode of DB Doctor identifies any stale data in the design.

Correct Mode
The correct mode cleans up stale data in design.

Running DB Doctor in the correct mode removes most of the stale data in design, but there might be some stale data that DB Doctor cannot correct. This gets reported in the Violation window. Such data includes connectivity issues that DB Doctor cannot automatically correct without impacting the netlist. Designers need to correct these Connectivity issues themselves.

Return Type

void

Syntax

```
sch::runDBDoctor -mode <modeName>
```

Parameters

Parameter	Type	Description
`ModeName`	`string`	Mode in which DB Doctor should run. Valid values are detect or correct. This parameter is required.

Examples

```
sch::runDBDoctor -mode detect
```

runDesignRules

Runs set of selected DRC rules on the design. DRC rules are set using Edit - Preferences - Project Preferences inside the Design Rule Checker tab.

Return Type

NONE

Syntax

```
runDesignRules
```

Examples

```
runDesignRules
```

runElectricalStress

This command runs electrical stress analysis on the design based on the setting specified in the 'Electrical Stress Settings' dialog.
Use the sdaReliability::electricalStressPreferences command to access the 'Electrical Stress Settings' dialog.

Return Type

NONE

Syntax

```
sdaReliability::runElectricalStress
```

Examples

```
sdaReliability::runElectricalStress
```

Related Commands

electricalStressPreferences

runMTBFAnalysis

Runs MTBF analysis on the design based on the settings specified in the 'Design MTBF Settings' dialog box.
Use the sdaReliability::MTBFPreferences command to access the 'Design MTBF Settings' dialog box.

Return Type

NONE

Syntax

```
sdaReliability::runMTBFAnalysis
```

Examples

```
sdaReliability::runMTBFAnalysis
```

Related Commands

sdaReliability::MTBFPreferences

runSchematicAudit

This command runs all the schematic audit rules specified in 'Schematic Audit Settings' dialog as error, warning, info or do not run.
sdaReliability::schematicAuditPreferences command can be used to access the Schematic Audit Settings dialog.

This command can also take specific list of rules to run and dump them in file specified in the command. If no file path is provided, audit run results will be shown in Audit dashboard only and no file will be dumped.

Return Type

NONE

Syntax

```
sdaReliability::runSchematicAudit ?<rulenames>? ?<filename>?
```

Parameters

Parameter	Type	Description
`rulenames`	`LIST`	Optional parameter specifying the list of rule names to be run. If this parameter is not specified, rules selected in the Schematic Audit Settings are run. This parameter is optional. Default value is `NULL`.
`filename`	`STRING`	Optional parameter specifying the location where the file is to be dumped. If this parameter is not specified, output file will not be dumped. This parameter is optional. Default value is `NULL`.

Examples

```
sdaReliability::runSchematicAudit
sdaReliability::runSchematicAudit {"unrecognisedDevice" "jedecMissing"}
sdaReliability::runSchematicAudit {"unrecognisedDevice" "jedecMissing"}
"../auditDump.csv"
```

Related Commands

schematicAuditPreferences

runThermalAnalysis

Runs thermal analysis on the design using the board file provided. If no board file path is provided, a dialog box will open to browse for the board file from the design.

Return Type

NONE

Syntax

```
sdaReliability::runThermalAnalysis <board_file_path>
```

Parameters

Parameter	Type	Description
`Board File Path`	`string`	Path of the board file of the design This parameter is optional. Default value is `NONE`.

Examples

```
sdaReliability::runThermalAnalysis
#Dialog will open to select a board file on which the thermal analysis will run.
sdaReliability::runThermalAnalysis
"D:/syscap_proj/Thermal_TV/output/ddr3/physical/Thermal_TV.brd"
#Thermal analysis will run on the Thermal_TV.brd file.
```

saveAll

Saves all the pages of the current project. Returns '0' if successful.

Return Type

INT

Syntax

```
saveAll
```

Examples

```
saveAll
```

Related Commands

undo

redo

saveDesign

calls cps::refreshProjectExplorer

Return Type

NONE

Syntax

```
saveDesign
```

Examples

```
saveDesign
```

saveScreenShot

Saves the screenshot of the application window in the specified png file.

Return Type

BOOLEAN

Syntax

```
saveScreenShot <file_path>
```

Parameters

Parameter	Type	Description
`<file_path>`	`STRING`	Path of the file. This parameter is required.

Examples

```
saveScreenShot P:/systemCaptureProjects/workshop1/workshop1.cpm
```

schematicAuditPreferences

This command opens the 'Schematic Audit Settings' dialog for setting the audit rules to be run on the design.

Return Type

NONE

Syntax

```
sdaReliability::schematicAuditPreferences
```

Examples

```
sdaReliability::schematicAuditPreferences
```

Related Commands

electricalStressPreferences

selectionFilter

It allows the elements to be selected on the canvas.

Return Type

NONE

Syntax

```
selectionFilter [list <list_of_elements>]
```

Parameters

Parameter	Type	Description
`[list <list_of_elements>]`	`LIST`	list of elements that will be allowed to be selected on canvas. For example: "Block Arrow" "Block Shape" "Bus" "Component" "Connect Line" "Drawing Object" "Navigation Links" "NetGroup" "Note" "Pin" "Property" "Signal Name" "Special Symbols" "System Design Objects" "Table" "Wire" This parameter is required.

Examples

```
#command to allow selection for all elements
selectionFilter [ list "All" ]
#command to allow selection for none:
selectionFilter [ list "" ]
#command to allow selection for few elements types
selectionFilter [ list "Connect Line" "System Design Objects" ]
```

selectItem

Selects and opens the item, either page or block, passed as itemId in Project Viewer.
viewType is a predefined string identifying the type of object to be opened.
itemType is a predefined string identifying the subtype of object to be opened.

The following values are supported for the viewType ItemType pairs:
SCH PAGE - To open a schematic page
FILE TEXT - To open a text file
START_PAGE START_PAGE - To open a Start Page for System Capture

Return Type

STRING

Syntax

```
selectItem <itemId> <viewType > <itemType>
```

Parameters

Parameter	Type	Description
`itemId`	`STRING`	Identifier for the object to be opened This parameter is required.
`viewType`	`STRING`	Predefined string identifying the type of object to be opened. This parameter is required.
`itemType`	`STRING`	Predefined string identifying the subtype of object to be opened. This parameter is required.

Examples

```
# Example - 1
# This command will return the itemId of the current active page of Schematic Editor
set activePage [sch::dbGetActivePageSpath]
puts $activePage
# If ref_5g sample project is opened, this command will return the page's itemId of the
current active page, example shown below
# @worklib.ref_5g(tbl_1):Audio(3)

# This command will select the page Audio(3) in Project Viewer
selectItem $activePage SCH PAGE

# Example - 2
# This command will return the itemId of the current active block of Schematic Editor.
set activeBlk [sch::dbGetSPathForActiveTab]
puts $activeBlk
# If ref_5g sample project is opened, this command will return the block's itemId of
currently active page, example shown below
# @worklib.ref_5g(tbl_1)

# This command will select the block blk1 in Project Viewer
selectItem $activeBlk SCH BLOCK
```

Related Commands

dbGetActivePageSpath

dbGetSPathForActiveTab

selectObject

This command is available for schematic pages and symbol views.

For schematic pages, the command selects the specified item(s) on the canvas. If incorrect arguments are provided, nothing is selected. For symbol views, the command selects objects on the symbol canvas, such as symbol outline, pins, pin properties, text, notes etc. You can also select multiple objects using this command.

Return Type

NONE

Syntax

```
selectObject ?-occPath <pagePath>? ?-type <objectType>? ?+? <xCoordinate> <yCoordinate>
| <xCoordinate> <yCoordinate> | area ?+? <xCoordinateTopLeft> <yCoordinateTopLeft>
<xCoordinateBottomRight> <yCoordinateBottomRight> | all | none | ?-pg <pageId>? -type
<objectType> -name <objectName> | -name <objectName> -pin <pinName> | -type term -pinid
[list <pinName1> <pinName2> ...] ?-fitInView? | -typeL [list <objectType1>
<objectType2> ...] -nameL [list <objectName1> <objectName2> ...] -pinL [list <pinName1>
<pinName2> ...]
```

Parameters

Parameter	Type	Description
`-occPath`	`STRING`	Schematic page path to perform selection on. This parameter is optional. Default value is `Currently open schematic page`.
`-type`	`STRING`	Object type. Applies to schematic page and symbols objects. This parameter is optional. Default value is `None`.

`xCoordinate`	`INT`	X coordinate of the selection point. Applies to schematic pages. Denotes incremental selection when prefixed with a '+'. This parameter is optional. Default value is `None`.
`yCoordinate`	`INT`	Y coordinate of the selection point. Applies to schematic pages. This parameter is optional. Default value is `None`.
`area <xCoordinateTopLeft> <yCoordinateTopLeft> <xCoordinateBottomRight> <yCoordinateBottomRight>`	`INT`	Selects all objects within the rectangle defined by the left, top, right, bottom coordinates. Denotes incremental selection when the left coordinate is prefixed with a '+'. This parameter is optional.
`all`	`STRING`	Selects all items on the currently open schematic page. This parameter is required.
`none`	`STRING`	Deselects all items on the currently open schematic page. This parameter is required.
`-pg`	`STRING`	Page id of the symbol tab. When page name is not provided, selectObject works on the active page. This parameter is optional. Default value is `None`.
`-name`	`STRING`	Name of the object to be select. Applies to symbols. This parameter is required.

`-pin`	`STRING`	Name of the pin to select. The type parameter should be passed the value 'TERM'. Applies to symbols. This parameter is required.
`-pinid`	`LIST`	List of names of the symbol pins to be selected. This parameter is required.
`-typeL`	`LIST`	List of object types to be selected. Applies to symbols. This parameter is required.
`-nameL`	`LIST`	List of names of objects to be selected. Applies to symbols. This parameter is required.

Examples

```
# Examples applicable to schematic pages
# Selects the component instance (objectType INST) at 9779 12464 on the page identified
by @worklib.ref_5g(tbl_1):Cam(17)
selectObject -occPath @worklib.ref_5g(tbl_1):Cam(17) -type INST 9779 12464

# Selects all objects at point
selectObject 4547 13395

# Selects all objects within a rectangle
selectObject area 2657 7698 13286 13094

# Selects all objects at a point and adds them to the existing set of selected objects
selectObject -occPath @worklib.ref_5g(tbl_1):Cam(17) -type INST + 10004 6864

# Selects all objects within a rectangle and adds them to the existing set of selected
objects
selectObject area + 5915 2373 9858 7813

# Examples applicable to symbol views
#Select symbol note
selectObject -type NOTE -name resistor
```

```
#Select symbol property
selectObject -type PROP -name SYM_PROP

#Select pin
selectObject -type term -name PIN_A

#Select pin notes and properties
selectObject -typeL [list {NOTE} {PROP} {PROP}] -nameL [list {EARLY_A} {HA1} {$PN}] -
pinL [list {EARLY_A} {HA1} {HA5}]

#Select pin note
selectObject -type NOTE -name pinA -pin {PIN_A}

#Select pin property
selectObject -type PROP -name PIN_PROP -pin {PIN_A}

#Variants of selectObject command for pins
selectObject -type term -pinid [list {-48V_B} ] -fitInView

selectObject -type term -pinid [list {-48V_B} ]

# This command selects all the items on the canvas
selectObject all

# Below command will remove selection from all the selected items on canvas.
selectObject none
```

selectTable

Select the row and column of a table on the schematic page.

Return Type

INT

Syntax

```
selectTable -col <col_num>| -row <row_num> | -cell <topleft_row_num> <topleft_col_num>
<bottomRight_row_num> <bottomRight_col_num
```

Parameters

Parameter	Type	Description
`col_num`	`INT`	The column number This parameter is required.
`row_num`	`INT`	The row number This parameter is required.
`-cell <topleft_row_num> <topleft_col_num> <bottomRight_row_num> <bottomRight_col_num>`	`INT`	The top-left and bottom-Right row and column number. This parameter is required.

setActionEnabled

Enables or disables an action associated with a menu or toolbar item.

Return Type

INT

Syntax

```
cps::setActionEnabled <context> <actionId> <enabled>
```

Parameters

Parameter	Type	Description
`context`	`STRING`	The context in which this action is supported. For schematic canvas actions, this should be "sch" This parameter is required.
`actionId`	`INT`	Numeric identifier of an action This parameter is required.
`enabled`	`BOOL`	1 enables the action, 0 disables the action This parameter is required.

Examples

```
# Disable print in schematic context
set printActionId [getActionId sch "Print"]
setActionEnabled sch $printActionId 0
```

Related Commands

getActionId

getActionName

setActionEnabledInNonElectricMode

setActionEnabledInReadOnlyState

setActionEnabledInNonElectricMode

Menus related to drawing electrical items like Drawing Wire, Bus, etc are not enabled when TOC pages are opened. To enable such menus for TOC pages setActionEnabledInNonElectricMode command is used.
TOC pages are opened in Schematic context, thus "sch" as context is passed.

Return Type

BOOL

Syntax

```
setActionEnabledInNonElectricMode <context> <actionId>
```

Parameters

Parameter	Type	Description
`context`	`STRING`	Context in which action was created. For schematic actions, context is "sch" This parameter is required.
`actionId`	`INT`	ID of action which needs to be enabled for TOC pages. This parameter is required.

Examples

```
# Add a single action directly to the top-level "Custom Menu"
set customAction4 {Custom Action 4}
addActionToMenu $customMenu $customAction4 ::customMenus::action4
$::customMenus::customIcon {Custom Menu - Action 4} {} {sch}

# Enable action4 in non-electric mode
set actionId4 [getActionId $context $customAction4]
setActionEnabledInNonElectricMode $context $actionId4
```

Related Commands

isActionEnabledInReadOnlyState

setActionEnabledInReadOnlyState

By default, menus are disabled if a block is read-only. Use this command to enable the menus for blocks in the read-only state.

Return Type

BOOL

Syntax

```
setActionEnabledInReadOnlyState <context> <actionId>
```

Parameters

Parameter	Type	Description
`context`	`STRING`	Context in which action was created. For schematic actions, context is "sch" This parameter is required.
`actionId`	`INT`	ID of action This parameter is required.

Examples

```
# Add a single action directly to the top-level "Custom Menu"
set customAction4 {Custom Action 4}
addActionToMenu $customMenu $customAction4 ::customMenus::action4
$::customMenus::customIcon {Custom Menu - Action 4} {} {sch}

# Enable action4 in read-only mode
set actionId4 [getActionId $context $customAction4]
setActionEnabledInReadOnlyState $context $actionId4
```

Related Commands

isActionEnabledInReadOnlyState

setAppTitle

Customizes the application title. The title can be reset or text can be appended to an existing title.
if append = 0, the title will be reset to a new value
if append = 1, the new value will be appended to the existing title

Return Type

STRING

Syntax

```
cps::setAppTitle <append> <appTitle>
```

Parameters

Parameter	Type	Description
`append`	`INT`	A value of 0 overwrites the existing title. 1 appends to the existing title This parameter is required.
`appTitle`	`STRING`	The new application title string This parameter is required.

Examples

```
# If the existing title is "Allegro System Capture" and this needs to be overwritten,
use the following syntax
cps::setAppTitle 0 "New App Title"
# The title would be set to "New App Title"

# To append to the application title, pass 1 in the append parameter. For example, if
the current title is "Allegro System Capture", running this command:
cps::setAppTitle 1 " - Scripting Mode"
# would set the application title to "Allegro System Capture - Scripting Mode"
```

Related Commands

getAppTitle

setAutoPan

It toggles the auto-panning feature on the schematic page.

Return Type

NONE

Syntax

```
setAutoPan <choice>
```

Parameters

Parameter	Type	Description
`choice`	`BOOLEAN`	Auto-panning on/off This parameter is required.

Examples

```
#command to disable the auto pan for activities, such as, drawing a wire,
#on the schematic page:
setAutoPan off
#command to enable the auto pan for activities. such as drawing a wire,
#on the schematic page:
setAutoPan on
```

setAutoSave

It controls the automatic saving of the design.

Return Type

NONE

Syntax

```
setAutoSave <choice>
```

Parameters

Parameter	Type	Description
`choice`	`BOOLEAN`	Auto-save on/off This parameter is required.

Examples

```
#command to enable the autosave feature of the schematic page.
setAutoSave true
#command to disable the autosave feature of the schematic page.
setAutoSave false
```

setDEHDLImportColorMapping

Changes the specified color for DE-HDL designs being imported.

Return Type

BOOL

Syntax

```
setDEHDLImportColorMapping <dehdlColor> <rgb_color_code_list>
```

Parameters

Parameter	Type	Description
`dehdlColor`	`STRING`	The color name This parameter is required.
`colorCodeList`	`INTLIST`	The RGB color code in string list format This parameter is required.

Examples

```
#Overwrites color definitions
#The following command overwrites the default red color to Blush Red.
setDEHDLImportColorMapping red {229, 110, 148}
#The following command overwrites the Green color to Magenta.
setDEHDLImportColorMapping GREEN {0 255 255}
```

Related Commands

getDEHDLImportColorMapping

getDEHDLColorNames

setDirectiveValue

Sets the value of a project directive. It adds a new directive if it is not already present in the project.

Return Type

BOOL

Syntax

```
cps::setDirectiveValue <sectionName> <directiveName> <directiveValue> <valueType>
```

Parameters

Parameter	Type	Description
`sectionName`	`STRING`	Name of the project section This parameter is required.
`directiveName`	`STRING`	Name of the directive This parameter is required.
`directiveValue`	`STRING`	Value of the directive This parameter is required.
`valueType`	`STRING`	Type of the value. Allowed types are "STRING", "INT", "LONG", "DOUBLE", "BOOL" This parameter is required.

Examples

```
cps::setDirectiveValue "GLOBAL" "HOST_UID" "887987_X5" STRING
```

setDockWidgetFloating

This command sets the docked widget in a floating state. If widget is already floating no changes will be seen. If the widget is in minimized state, it changes to the floating state.

Return Type

BOOL

Syntax

```
cps::setDockWidgetFloating <dockWidgetName> <isFloating>
```

Parameters

Parameter	Type	Description
`dockWidgetName`	`STRING`	Name of the widget whose state needs to be floating This parameter is required.
`isFloating`	`BOOL`	If the docked widget needs to be set in floating state. This parameter is required.

Examples

```
# Following command will set the Command Window dock widget in floating state
cps::setDockWidgetFloating CP_TCL_WINDOW_DOCK true
```

Related Commands

createDock

addDock

setDockWidgetUndockDefaultSize

This command helps in setting the height and width of floating or undocked widget whose widget name is passed. Height and width are not applied to floating widgets.

If isRestored is set true, resizing of widget is not allowed for floating widgets.

Return Type

INT

Syntax

```
cps::setDockWidgetUndockDefaultSize <widgetName><width><height><isRestored>
```

Parameters

Parameter	Type	Description
`widgetName`	`STRING`	Name of the widget to be resized This parameter is required.
`width`	`INT`	Width of the widget This parameter is required.
`height`	`INT`	Height of the widget This parameter is required.
`isRestored`	`BOOL`	If default size of the widget to be restored or not This parameter is required.

Examples

```
# This command sets the following height and width of the Command Window.
cps::setDockWidgetUndockDefaultSize CP_TCL_WINDOW_DOCK 5 5 false
```

Related Commands

createDock

addDock

setDockedWidgetVisibility

setDockedWidgetVisibilityOff

setPropertiesWidgetVisibility

setErrorViolationWindowVisibility

Opens the Violation window.

Return Type

NONE

Syntax

```
cps::setErrorViolationWindowVisibility
```

Examples

```
cps::setErrorViolationWindowVisibility
```

setFillColor

It changes the fill color of the currently selected object on the canvas. The selected object has to be an enclosed shape for it to work correctly. It works for multiple selected objects as well.

Return Type

NONE

Syntax

```
setFillColor <color>
```

Parameters

Parameter	Type	Description
`color`	`COLOR`	Specify color name or a hexadecimal value for the color to fill This parameter is required.

Examples

```
setFillColor #f8d971
#this command changes the fill color of the selected object to #f8d971,
#which is the hexadecimal code for color yellow. To convert any color to its
#equivalent hexadecimal code, use any online converter.
setFillColor red
#this command changes the fill color of the selected object to red.
```

Related Commands

setFill

setLineColor

setHeaderFillColor

setTextColor

setBackgroundColor

setLogWindowVisibility

Opens the Session Log window.

Return Type

NONE

Syntax

```
cps::setLogWindowVisibility
```

Examples

```
cps::setLogWindowVisibility
```

setMonochromePrintingThreshold

This command helps in customizing the threshold value to enhance a black and white image.

In a black and white image, white pixels are represented by the pixels of the image whose value is greater than the threshold, and black pixels are
represented by pixels with values less than the specified threshold.

If you want an image to be darker then set a higher value of the threshold and for a brighter image, set the threshold value lower than the pixels value.

This command is an alternative for the MONOCHROME_PRINTING_THRESHOLD directive.

Return Type

BOOL

Syntax

```
sch::setMonochromePrintingThreshold <thresholdValue>
```

Parameters

Parameter	Type	Description
`thresholdValue`	`INT`	This value helps in either darkening or brightening a black and white image. Range of threshold value can be between 0-255, default value in System Capture is 127. This parameter is required.

Examples

```
# This command will set the threshold value to 180 which is greater than the default
value of 127 and will darken the image.
sch::setMonochromePrintingThreshold 180
```

setSearchWindowVisibility

Opens the Search window.

Return Type

NONE

Syntax

```
cps::setSearchWindowVisibility
```

Examples

```
cps::setSearchWindowVisibility
```

setShowMoveGuide

Toggles between showing and hiding guide lines.

Return Type

BOOLEAN

Syntax

```
setShowMoveGuide <choice>
```

Parameters

Parameter	Type	Description
`choice`		This parameter is required.

setStyles

Added functionality for styling and formatting for objects in single transaction through a Tcl script. This command is used to set various styles of selected items in a single transaction.

Return Type

void

Syntax

```
void setStyles(std::vector<std::string> args)
```

Examples

```
#Command to takes a vector parameter and used this vector to set styles such as
#line width, line type, line color, fill color, text font, text size, and so on in a
single transaction.
#select group items to be set and make a vector vec with following data
vec.push("setLineWidth")
vec.push( $lineWidth)
vec.push("setLineType")
vec.push( $lineStyle)
sch::setStyles $vec
```

setTclWindowVisibility

Opens the Tcl command window.

Return Type

NONE

Syntax

```
cps::setTclWindowVisibility
```

Examples

```
cps::setTclWindowVisibility
```

setTreeData

This command sets tree data for the tree with specified handle.

Return Type

INT

Syntax

```
cps::setTreeData <treeHandle> <data>
```

Parameters

Parameter	Type	Description
`treeHandle`	`handler`	Handle of the tree node This parameter is required.
`data`	`treedata`	Data model for tree node This parameter is required.

Examples

```
#Command to create the tree view using the following command
#and store the handle in the variable called nSymbolTree
set lSymbolTree [ cps::createSCTreeView ]

#Get the tree data that needs to be set
set lTreeData [ cps::getSCComponentExplorerData "" ]

#Now set the tree data on the specified tree handle
cps::setTreeData $lSymbolTree $lTreeData
```

setTreeNodeDisplayString

This command will set the display name of the specified tree node.

Return Type

NONE

Syntax

```
cps::setTreeNodeDisplayString <treeHandle> <treeNode><displayName>
```

Parameters

Parameter	Type	Description
`displayName`	`STRING`	Display name for the tree node This parameter is required.
`treeNode`	`treedata`	Node handler on which display name needs to be set This parameter is required.
`treeHandle`	`Handle`	Handle of the tree in which node is present This parameter is required.

Examples

```
#Command to store the tree handler in the variable called nProjectExplorerParentNode
#for CP_PROJECT_EXPLORER and then create a node named - PSpiceDesign and store
#the handle of this node into variable called designNode

variable nProjectExplorerParentNode ""
set nProjectExplorerParentNode [ cps::getTree CP_PROJECT_EXPLORER ]

set designNode [cps::createTreeNode $nProjectExplorerParentNode [ cps::getTreeRootData
$nProjectExplorerParentNode ] "" PSpiceDesign 0 ]

#Now set the display name of the string:
cps::setTreeNodeDisplayString "ABC"
```

setTreeNodeIcon

This command will set the icon for the particular node. In the UI, the icon will be visible prior to the node name.

Return Type

NONE

Syntax

```
cps::setTreeNodeIcon <treeHandle> <treeNode> <iconPath>
```

Parameters

Parameter	Type	Description
`<treeHandle>`	`Handle`	Handle of the tree in which node is present This parameter is required.
`<treeNode>`	`Handle`	Node handler on which icon needs to be set This parameter is required.
`<iconPath>`	`String`	Path of the desired icon This parameter is required.

Examples

```
#Command to store the tree handler in the variable called nProjectExplorerParentNode
#for CP_PROJECT_EXPLORER and then create a node named - PSpiceDesign and store
#the handle of this node into variable called designNode

variable nProjectExplorerParentNode ""
set nProjectExplorerParentNode [ cps::getTree CP_PROJECT_EXPLORER ]

set designNode [cps::createTreeNode $nProjectExplorerParentNode [ cps::getTreeRootData
$nProjectExplorerParentNode ] "" PSpiceDesign 0]

#Now set the icon for this particular node
cps::setTreeNodeIcon $nProjectExplorerParentNode $designNode "Folder_default.png"
```

setTreeNodeItemType

This command will set the type of the give node. The content on clicking the node will be treated as TEXT if the item type is set to TEXT.

Return Type

NONE

Syntax

```
cps::setTreeNodeItemType <treeNode> <viewType>
```

Parameters

Parameter	Type	Description
`<treeNode>`	`Handler`	Node handler on which item type needs to be set This parameter is required.
`<viewType>`	`STRING`	Item Type. example- TEXT This parameter is required.

Examples

```
#Command to store the tree handler in the variable called nProjectExplorerParentNode
#for CP_PROJECT_EXPLORER and then create a node named - MyDesign and store the
#handle of this node into variable called designNode

variable nProjectExplorerParentNode ""
set nProjectExplorerParentNode [ cps::getTree CP_PROJECT_EXPLORER ]

set designNode [cps::createTreeNode $nProjectExplorerParentNode [ cps::getTreeRootData
$nProjectExplorerParentNode ] "" MyDesign 0]

#Now set the node item type
cps::setTreeNodeItemType $designNode "TEXT"
```

Related Commands

setTreeNodeViewType

setTreeNodeMenuID

This command will set the menu over the corresponding tree node. RMB on the node will show the menu defined under menuId.

Return Type

NONE

Syntax

```
cps::setTreeNodeMenuID <treeHandle> <treeNode> <menuId>
```

Parameters

Parameter	Type	Description
`<treeHandle>`	`Handler`	Handle of the tree in which node is present This parameter is required.
`<treeNode>`	`Handler`	Node handler on which menu needs to be set This parameter is required.
`<menuId>`	`INT`	ID of the menu which need to be shown on RMB over the node This parameter is required.

Examples

```
#Command to store the tree handler in the variable called nProjectExplorerParentNode
#for CP_PROJECT_EXPLORER and then create a node named - MyDesign and store the
#handle of this node into variable called designNode

variable nProjectExplorerParentNode ""
set nProjectExplorerParentNode [ cps::getTree CP_PROJECT_EXPLORER ]

set designNode [cps::createTreeNode $nProjectExplorerParentNode [ cps::getTreeRootData
$nProjectExplorerParentNode ] "" MyDesign 0]

set menuID 7778
#Now set the menu ID
cps::setTreeNodeMenuID $nProjectExplorerParentNode $designNode $menuID
```

setTreeNodeOpenCommand

This command will set the command with the tree node. Clicking the node will execute the set command.

Return Type

NONE

Syntax

```
cps::setTreeNodeOpenCommand <treeNode> <openCommand>
```

Parameters

Parameter	Type	Description
`<treeNode>`	`Handle`	Node handler on which open command needs to be set This parameter is required.
`<openCommand>`	`STRING`	This is the name of the command which will run on click of the node This parameter is required.

Examples

```
#Command to store the tree handler in the variable called nProjectExplorerParentNode
#for CP_PROJECT_EXPLORER and then create a node named - PSpiceDesign and store the
#handle of this node into variable called designNode

variable nProjectExplorerParentNode ""
set nProjectExplorerParentNode [ cps::getTree CP_PROJECT_EXPLORER ]

set designNode [ cps::createTreeNode $nProjectExplorerParentNode [ cps::getTreeRootData
$nProjectExplorerParentNode ] "" PSpiceDesign 0 ]

#Create the child node of the designNode
set childNodePtr [ cps::createTreeNode $nProjectExplorerParentNode $designNode ""
ChildNode 0 ]

#Now, set the tree node open command:
cps::setTreeNodeOpenCommand $childNodePtr "openItem"
```

setTreeNodeViewType

This command will set the view type of the given node. By setting the view type to FILE, click on node will be treated as the file.

Return Type

NONE

Syntax

```
cps::setTreeNodeViewType <treeNode> <viewType>
```

Parameters

Parameter	Type	Description
`<treeNode>`	`Handler`	Node handler on which item type needs to be set This parameter is required.
`<viewType>`	`STING`	view type of the node This parameter is required.

Examples

```
#Store the tree handler in the variable called nProjectExplorerParentNode for
#CP_PROJECT_EXPLORER and then create a node named - MyDesign and store
#the handle of this node into variable called designNode

variable nProjectExplorerParentNode ""
set nProjectExplorerParentNode [ cps::getTree CP_PROJECT_EXPLORER ]

set designNode [cps::createTreeNode $nProjectExplorerParentNode [ cps::getTreeRootData
$nProjectExplorerParentNode ] "" MyDesign 0]

#Now set the node view type
cps::setTreeNodeViewType $designNode "FILE"
```

Related Commands

setTreeNodeItemType

setVisibility

It sets the visibility for the provided dialog handle. With the visibility value of 1, the dialog becomes visible. With the visibility value of 0, the dialog gets hidden.

Return Type

VOID

Syntax

```
sdaUI::setVisibility <dialogHandle> <visibility>
```

Parameters

Parameter	Type	Description
`dialogHandle`	`STRING`	Handle of dialog This parameter is required.
`visibility`	`BOOL`	Visibility value This parameter is required.

Examples

```
sdaUI::hybridDialog {myblog} {http://myblog.com} {blogname}
#output: dialogHandle 60 hybridHandle 57
#command to use dialog handle
sdaUI::setVisibility 60 1
```

Related Commands

hybridDialog

showBitNumbers

Show or hide the bit number display on selected bus taps on canvas

Return Type

NONE

Syntax

```
sch::showBitNumbers <bState>
```

Parameters

Parameter	Type	Description
`<bState>`	`BOOL`	Value to be set to show or hide the bit number display This parameter is required.

Examples

```
To show the bit numbers on the selected bus taps on the canvas-
sch::showBitNumbers 1

To hide the bit numbers on the selected bus taps on the canvas-
sch::showBitNumbers 0
```

showPhysicalNetNames

It shows the physical net names of all the nets in the design of all the open pages in the application.

Return Type

NONE

Syntax

```
showPhysicalNetNames
```

Examples

```
showPhysicalNetNames
```

showToolBar

It will display the top most toolBar.

Return Type

NONE

Syntax

```
showToolBar
```

Examples

```
::showToolBar
```

Related Commands

hideToolBar

showWorkFlow

This Tcl command is used to launch the In-Design Workflow widget in System Capture. It is launched only when Allegro EDM Flow Manager is running.

Return Type

None

Syntax

```
cpb::showWorkFlow
```

Examples

```
cpb::showWorkFlow
```

storeAttachment

It stores a file in the currently opened System Capture design as an attachment. The file path can be absolute or relative to the project. On successful execution of the command, the file gets stored as an attachment in the currently opened design. It also creates a backup copy of the design before storing the file as an attachment. The backup path is specified in the return of the command. If the path of the file to be attached is wrongly specified, the command reports the missing file error.

Return Type

STRING

Syntax

```
sch::storeAttachment <filePath>
```

Parameters

Parameter	Type	Description
`filePath`	`STRING`	name of the file to store as an attachment in design This parameter is required.

Examples

```
#command to store the test.png as attachment in the currently opened design.
sch::storeAttachment {d:/test.png}
#output: 1 file(s) successfully added to System Capture database. Backup of the
#old System Capture database created at:
#D:/testcases/174/testProj/top/logic/top.sdax.1565686965.bak
#command to report failure as the file is not present at the given location
sch::storeAttachment {d:/test1.png}
#output: Error:Could not attach the following file d:/test1.png to System Capture
database.
```

Related Commands

deleteAllAttachments

deleteAttachment

getAllAttachments

getAttachment

listAttachments

tocNumberingMode

Changes the numbering mode of the Table Of Contents (TOC) pages in TOC section of the specified design block.

Return Type

BOOLEAN

Syntax

```
tocNumberingMode <design_name> <numbering_mode>
```

Parameters

Parameter	Type	Description
`<design_name>`	`STRING`	Name of the design block. This parameter is required.
`<numbering_mode>`	`STRING`	Mode of numbering whether decimal or roman. This parameter is required.

Examples

```
tocNumberingMode workshop1 roman
```

undo

Reverses the action of an earlier action. You can reverse multiple actions done during a session until the design is saved.

Return Type

BOOLEAN

Syntax

```
undo
```

Examples

```
undo
```

unroute

Converts the selected route segment to rat.

Return Type

NONE

Syntax

```
unroute
```

Examples

```
unroute
```

updateAll

Updates the logical design with the differences between the board and the logical design.

Return Type

NONE

Syntax

```
updateAll
```

Examples

```
updateAll
```

updateConstraints

Updates the logical design with the constraint differences between the logical design and the board design.

Return Type

NONE

Syntax

```
updateConstraints
```

Examples

```
updateConstraints
```

useLibs

It updates the Project Libraries list in the Project Preferences dialog.

Return Type

NONE

Syntax

```
useLibs <library_name>
```

Parameters

Parameter	Type	Description
`library_name`	`STRING`	Name of the library This parameter is required.

Examples

```
#command to add 'aa' to the list of Libraries in the Project Preferences dialog.
useLibs aa
```

viewThermalMap

Runs Celcius 2D for the currently selected temperature in the thermal dashboard.

Return Type

NONE

Syntax

```
sdaReliability::viewThermalMap <temperature>
```

Parameters

Parameter	Type	Description
`temperature`	`INT`	Temperature value This parameter is required.

Examples

```
sdaReliability::viewThermalMap "27"
#Command will launch Celcius 2D UI for the selected board at 27 degrees Celcius
```

Related Commands

sdaReliability::runThermalAnalysis

zoom

Zooms in or out on the canvas.

Return Type

NONE

Syntax

```
zoom -<type>
```

Parameters

Parameter	Type	Description
`type`	`STRING`	value will be "-in" for zoom in and "-out" for zoom out This parameter is required.

Examples

```
zoom -in
```

8

Symbol Editor Tools

add

Use this command to add a text note on the symbol page or a property to the symbol. The value of the second parameter determines the function performed by the command.

Return Type

null

Syntax

```
add ?[-pg <page_name>]? -richnote -pos <position> -value <new_text> | -prop -name
<propname> -value <propvalue>
```

Parameters

Parameter	Type	Description
`-pg <page_name>`	STRING	Optional parameter specifying the page where the note is to be added. If this parameter is not specified, the text note is added to the current page. This parameter is optional.
`-richnote`	KEYWORD	Use this keyword to add text note on the symbol page. This parameter is required.
`-pos <position>`	LIST	The X-Y coordinate of the top-left corner of the note item. Coordinates are represented as a list of two integers representing X, Y. This parameter is required.
`-value <new_text>`	STRING	The text note to be added. This parameter is required.
`-prop`	KEYWORD	Use this keyword to add a property to the symbol. This parameter is required.
`-name <propname>`	STRING	Name of the property to be attached to the symbol. This parameter is required.
`-value <propvalue>`	STRING	Value to be assigned to the symbol property. This parameter is required.

Examples

```
#command to add the note "This is test symbol" to the symbol starting from X-Y
coordinate 5580, -950
add -richnote -value {testnote} -pos [list 5580 -950]

#command to add the 'myversion' property with value '2.0' to the symbol on the current
page
add -prop -name myversion -value 2.0;
```

Related Commands

edit

addArc

This command adds an arc on the symbol canvas using start angle, end angle and the rectangle coordinates enclosing the full circle arc.

Return Type

NULL

Syntax

```
addArc ?-pg <page_name>? -start <start angle> -end <end angle> -rect <topLeft X
coordinate> <topLeft Y coordinate> <bottomRight X coordinate> <bottomRight Y
coordinate>
```

Parameters

Parameter	Type	Description
`-pg <page_name>`	`STRING`	Optional parameter used to specify the symbol page where arc is to be added. If no value is specified, arc is added to the current page. This parameter is optional. Default value is `Current page`.
`-start <start angle>`	`DOUBLE`	Start angle of the arc. This parameter is required.
`-end <end angle>`	`DOUBLE`	End angle of the arc. This parameter is required.
`-rect <topLeft X coordinate>`	`DOUBLE`	X coordinate of the top left corner of rectangle enclosing the arc. This parameter is required.
`<topLeft Y coordinate>`	`DOUBLE`	Y coordinate of the top left corner of rectangle enclosing the arc. This parameter is required.
`<bottomRight X coordinate>`	`DOUBLE`	X coordinate of the bottom right corner of rectangle enclosing the arc. This parameter is required.
`<bottomRight Y coordinate>`	`DOUBLE`	Y coordinate of the bottom right corner of rectangle enclosing the arc. This parameter is required.

Examples

```
addArc -start 0 -end 180 -rect 0 -75 250 175
```

Related Commands

addLine

addBlock

addBlock

Draws a block of the specified shape on the symbol canvas.

Return Type

Return

Syntax

```
addBlock <block shape name> ?-pg <page_name>? -rect [list <topLeft x-coordinate>
<topLeft y-coordinate>] [list <bottomRight x-coordinate> <bottomRight y-coordinate>]
```

Parameters

Parameter	Type	Description
`-pg <page_name>`	`STRING`	Name of the symbol page to which the block shape is to be added. This parameter is optional. Default value is `Current Page`.
`<block shape name>`	`STRING`	Block shape to be added. Supported block shapes are: BlockRect and BlockOval. This parameter is required.
`-rect [list <topLeft x-coordinate> <topLeft y-coordinate>] [list <bottomRight x-coordinate> <bottomRight y-coordinate>]`	`INT`	Rectangle coordinates (Top left X-Y coordinate, Bottom right X-Y coordinate) This parameter is required.

Examples

```
#Command to create a circle with page name
addBlock BlockOval -pg
{{"cellID":"ci","keyType":"view","libID":"worklib","viewID":"sym_1"}} -rect [list -410
-1020] [list -180 -740]

#Command to create a rectangle with page name
addBlock BlockRect -pg
{{"cellID":"ci","keyType":"view","libID":"worklib","viewID":"sym_1"}} -rect [list -730
-1090] [list -470 -780]

#Command to create a circle without page name
addBlock BlockOval -rect [list -410 -1020] [list -180 -740]

#Command to create a rectangle without page name
addBlock BlockRect -rect [list -730 -1090] [list -470 -780]
```

Related Commands

addLine

addArc

addLine

Draws a line on the symbol canvas using the grid coordinates specified in the command.

Return Type

None

Syntax

```
addLine ?-pg <page_name>? <start point x-coordinate> <start point y-coordinate> <end
point x-coordinate> <end point y-coordinate>
```

Parameters

Parameter	Type	Description
`-pg <page_name>`	`STRING`	Name of the page to which line is to be added. This parameter is optional. Default value is `Current Page`.
`<start point x-coordinate>`	`INT`	X coordinate of the first end point. This parameter is required.
`<start point y-coordinate>`	`INT`	Y coordinate of the first end point. This parameter is required.
`<end point x-coordinate>`	`INT`	X coordinate of the second end point. This parameter is required.
`<end point y-coordinate>`	`INT`	Y coordinate of the second end point. This parameter is required.

Examples

```
#Adds line on the canvas
addLine -950 1050 -210 750
```

Related Commands

addArc

addBlock

edit

This command is used to modify the value of a property assigned to a symbol, or to modify pin properties such as pin name, pin number, pin type, and values assigned to the pin properties.

Return Type

None

Syntax

```
edit ?-pg <page_name>? -pin <pin_name> -type<new_pin_type> | -note <pin_note> -value
<new_value> | -prop <prop_name> -section <section_number> -propval <prop_value> -value
<new_value> | -prop <prop_name> -value <new_value>
```

Parameters

Parameter	Type	Description
`-pg <page_name>`	STRING	Symbol Page where the operation is to be performed. If this parameter is not used then current page is used as the default page. This parameter is optional.
`-pin <pin_name>`	STRING	Name of the pin. Use this parameter to modify pin properties. This parameter is required.
`-type<new_pin_type>`	STRING	New Type of the pin. Valid pin types are : Input, Output, Inout, Oc (Open Collector), Oc_inout, Oe (Open Emitter), Oe_inout, Ts, Ts_inout , Power, Ground, Nc (No Connect), Analog, Unspec This parameter is required.
`-note <pin_note>`	STRING	Display name of the pin This parameter is required.
`-value <new_value>`	STRING	New value to be assigned. Used in conjunction with 'note' and 'prop' parameters. This parameter is required.
`-prop <prop_name>`	STRING	Name of the property to be modified. For modifying pin properties, use this parameter with the 'pin' parameter. Without 'pin' parameter, specified symbol property is modified. This parameter is required.
`-section <section_number>`	STRING	Section containing the pin to be modified. This parameter is required.
`-propval <prop_value>`	STRING	Current property value that is to be modified. This parameter is required.

Examples

```
Command to modify the LOCATION property of the symbol and assign it the value A
edit -prop {$LOCATION} -value {A}

Command to modify the Pin Number property of the Pin B and assign it the value 2 in
section 1
edit -pin {B} -prop {$PN} -section {1} -propval {1} -value {2}

Command to modify the Pin Text ( Display Name) of the Pin A and assign it the value 2
in section 1. It is like rename Pin A to B.
edit -pin {A} -note {A} -value {B} -pinid {A} ;

Command to modify the Pin Type of the Pin A and assign it the type InOut. Valid pin
types are : Input, Output, Inout, Oc, Oc_inout, Oe, Oe_inout, Ts, Ts_inout , Power,
Ground, Nc, Analog, Unspec
edit -pin {A} -type {Inout} -pinid {A};
```

execCmd

This command is used for adding pin with all predefined parameters
Or
This command is used to edit the Pin Name of a pin.

Return Type

null

Syntax

```
execCmd addPin -pin ?[-pg <page_name>]? -name <pin_name> -type <pin_type> -side
<pin_side> -shape <pin_shape> | editPinName ?[-pg <page_name>]? -pin <pin_name> -name
<new_name>
```

Parameters

Parameter	Type	Description
`pin_name`	`STRING`	Name of the pin This parameter is required.
`pin_type`	`STRING`	It specifies Pin Type. It has 13 predefined types like Input, Output, Power, Ground, InOut This parameter is required.
`pin_side`	`STRING`	It specifies Pin Side. It has predefined side options Left, Right, Top, Bottom This parameter is required.
`shape`	`STRING`	It specifies shape for the pin. It has few predefined shapes like Line, Circle This parameter is required.
`auto`	`parameter`	It places pin on the outline automatically, otherwise pin will be placed on the center of symbol This parameter is required.

Examples

```
execCmd addPin -pin -name {YYjj} -type {Input} -side {Right} -shape {Line} -
nographicadjust -auto

execCmd editPinName -pin {B} -name {A} -autorename -pinid {B};
```

inlineEdit

Edits note and property values on symbols.

Return Type

None

Syntax

```
inlineEdit ?[-pg <page_name>]? -value <new_value>
```

Parameters

Parameter	Type	Description
`page_name`	`STRING`	Symbol Page where the operation is to be performed This parameter is optional.
`new_value`	`STRING`	New value of the edited object This parameter is required.

Examples

```
# Edit the value of a Note
selectObject -type NOTE -name note1
inlineEdit -value {note2}
```

Related Commands

selectObject

move

This command works only on pins. Select pin from pin table, by click or using selectObject command
1. Use -pos to move selected object to given x,y coordinates
2. Use -updateside to move object to any side of the symbol
possible values for updateside option are

Return Type

null

Syntax

```
move ? -pg <page_name> -pos <location_x location_y> | move ? -pg <page_name> -
updateside <pin_side>
```

Parameters

Parameter	Type	Description
`pg <page_name>`	`STRING`	The page on which object is present This parameter is optional. Default value is `Current tab`.
`pos <location_x location_y>`	`LIST`	The X-coordinate and Y-coordinate of the location at which the object is to be moved This parameter is required.
`-updateside <pin_side>`	`STRING`	The side to which the pin is moved related to symbol. The valid value are: This parameter is required.

Examples

```
1. move -pos 10750 7800
2. move -updateside {Left}
```

Related Commands

selectObject

moveTo

Use this command for moving pins from one block symbol to another. In case of primitive symbols, the command is used for moving pins from one section to another.

Return Type

Return

Syntax

```
execCmd moveTo ?-pg <page_name>? -pinids {<pin name>} {<pin name>} -nextsection | -
section <destination section> -cursymbols [list {<pin's source symbol number>} {pin's
source symbol number} ] -nextsymbol | -symbol <destination symbol number>
```

Parameters

Parameter	Type	Description
`-pg <page_name>`	`STRING`	Optional parameter to specify the symbol page where the operation is to be performed. If this parameter is not specified, current page is used by default. This parameter is optional.
`-pinids {<pin name>} {<pin name>}`	`STRING`	Name of the pins that are to be moved. This parameter is required.
`-nextsection`	`EMPTY`	Use this parameter when symbol pins are to be moved to a new section. This parameter is required.
`-section <destination section>`	`INT`	Use this parameter when symbol pins are to be moved to an existing section. The section number of the target section is specified as the parameter value. This parameter is required.
`-cursymbols [list {<pin's source symbol number>} {pin's source symbol number} ]`	`LIST`	Symbol number from where the pin is to be moved. This parameter is required.
`-nextsymbol`	`EMPTY`	Use this parameter when symbol pins are to be moved to a new block symbol. This parameter is required.
`-symbol<destination symbol>`	`INT`	Use this parameter when symbol pins are to be moved to an existing block symbol. The target symbol number is specified as the parameter value. This parameter is required.

Examples

```
#Moving pins from an existing section to a new section of the same symbol
execCmd moveTo -pinids {RCC_TX} {RX_IF} -nextsection

# Moving pins from one section to another of the same symbol
execCmd moveTo -pinids {RCC_TX} {RX_IF} -section 2

# Moving pins from one symbol to another - This is only valid for Block Symbols.
execCmd moveTo -pinids {RCC_TX} {RX_IF} -cursymbols [list {1} {1} ] -symbol 2
```

openItemSingleTab

This commands opens the requested item in single tab mode where per library single tab is used. If active library tab is in modified state with unsaved changes, requested item will open as a separate tab.

Return Type

Return

Syntax

```
openItemSingleTab <pagename to be opened> viewtype itemtype
```

Parameters

Parameter	Type	Description
`pagename`	`STRING`	Key of the view to be opened. This parameter is required.
`View type`	`STRING`	Type of view like SYMBOL, PARTVIEW etc. This parameter is required.
`Item type`	`STRING`	Type of item like SYM, PART, etc. This parameter is required.

Examples

```
//Open any symbol view
openItemSingleTab
{{"cellID":"con3","keyType":"view","libID":"axc","partID":"con3","viewID":"sym_1"}}
SYMBOL SYM

//Open consolidated schematic table view
openItemSingleTab
{{"cellID":"con3","keyType":"view","libID":"axc","partID":"con3","viewID":"GLOBAL"}}
SYMBOL SYM

//Open part view
openItemSingleTab {{"keyType":"part","libID":"axc","partID":"con3"}} PARTVIEW PART
```

Related Commands

openItem

starttransactionrecording

Use this command to mark the beginning of any operation or list of micro operations (wrapped inside a single operation) for which undo/redo is supported. The transactions created are based on the specified mode . The supported modes are:
1. MODE_CSDB : for symbol only operation
2. MODE_DDBPI : for chips only operation
3. MODE_BOTH : for symbol as well as chips operation

Return Type

Return

Syntax

```
execCmd starttransactionrecording ?-pg <page_name>? -mode <transaction_mode>;
```

Parameters

Parameter	Type	Description
`-pg <page_name>`	`STRING`	This is an optional parameter to specify the page name with the symbol or consolidated view where the command gets executed. If this parameter is not specified, by default command is run on the current page. This parameter is optional.
`-mode <transaction_mode>`	`STRING`	Valid transaction modes are: MODE_CSDB , MODE_DDBPI, and MODE_BOTH This parameter is required.

Examples

```
#Following command begins a transaction that is specifically for symbol
execCmd starttransactionrecording -pg
{{"cellID":"ci","keyType":"view","libID":"worklib","viewID":"sym_1"}} -mode
{MODE_CSDB};

#Following command begins a transaction that involves operation in symbol/chips.
execCmd starttransactionrecording -pg
{{"cellID":"ci","keyType":"view","libID":"worklib","viewID":"sym_1"}} -mode
{MODE_BOTH};
```

stoptransactionrecording

Use this command to commit the active transactions in process and mark them available for undo or redo.

Return Type

Return

Syntax

```
cpse::executeCommand stoptransactionrecording
```

Examples

```
cpse::executeCommand stoptransactionrecording
```

switchContext

For Symbol to annotate pin numbers from different sections, switchContext command can be used. This is used when same symbol is shared between different sections.
Symbol and pin table display the information for the specified section number. If any symbol is shared between primitives, then only -primitive option should be used

Return Type

null

Syntax

```
switchContext ?-pg <page_name> -section <section_number> ?-primitive<primitive_name>?
```

Parameters

Parameter	Type	Description
`-pg <page_name>`	`STRING`	Optional parameter specifying the page name of the symbol. If this parameter is not specified, default value is used. This parameter is optional. Default value is `Current tab name`.
`-section <section_number>`	`INT`	The section number to be displayed. This parameter is required.
`-primitive<primitive_name>`	`STRING`	Primitive name to open This parameter is optional. Default value is `Default primitive name, which is part_name property value`.

Examples

```
switchContext -section 2
```

9

System Level Design

dbGetBlocks

Returns the list of DBIDs of system blocks on a page

Return Type

LIST

Syntax

```
sch::dbGetBlocks <pageID>
```

Parameters

Parameter	Type	Description
`pageID`	`DBID`	The DBID of the page This parameter is required.

Examples

```
#Get the list of system blocks on the active Page
set blockIDList [ sch::dbGetBlocks [ sch::dbGetActivePage ] ]
```

Related Commands

dbGetBlockPins

dbGetMappedBlockName

dbGetMappedBlockType

dbGetItemName

dbGetConnectedBoardPins

Returns the list of pins connected to the net inside the block or board.
First input argument is DBID of block or board got from dbGetMappedBoardItem command.
Second input is net name whose connected pins are to be listed.

Return Type

LIST

Syntax

```
sch::dbGetConnectedBoardPins <boardItem> <netName>
```

Parameters

Parameter	Type	Description
`boardItem`	`STRING`	The DBID of hierarchical block or board This parameter is required.
`netName`	`STRING`	The name of the net in the block or board This parameter is required.

Examples

```
#Get the connected pin list for the net 'ADR0' inside block 'data_buffer'
sch::dbGetConnectedBoardPins [sch::dbGetMappedBoardItem data_buffer] ADR0

Output - {adr0 20 UNSPEC @worklib.data_buffer(tbl_1):\\I1\\:\\M8\\ ADR0 {} U1
@worklib.data_buffer(tbl_1):\\I1\\}
The output shows a pin having following values -
adr0 - Pin name
20 - Pin number
UNSPEC - Pin Direction
@worklib.data_buffer(tbl_1):\\I1\\:\\M8\\ - Pin SPath
ADR0 - Net Name
{} - Net SPath
U1 - RefDes
@worklib.data_buffer(tbl_1):\\I1\\ - Section SPath
```

Related Commands

dbGetMappedBoardItem

dbGetConnectedInstTerms

Returns a list of DBIDs of pins connected to a netBitContext which is in-context object for a net bit.

Return Type

LIST

Syntax

```
sch::dbGetConnectedInstTerms <netBitContext>
```

Parameters

Parameter	Type	Description
`netBitContext`	`DBID`	The DBID of the in-context net bit object This parameter is required.

Examples

```
# after selecting a NetGroup on the page and running the following sequence of commands
returns the list of netBitContext DBIDs of members of the NetGroup
#get the pageId of the active page
set pgId [sch::dbGetActivePage]
#get the list of dbIDs of all the selected NetGroups on the page
set selectedNetGroupIds [sch::dbGetSelectedItems $pgId]
#get the dbID of the first selected pin in the list of pins' dbIDs
set firstSelectedNetGroupId [lindex $selectedNetGroupIds 0]
#get the data of the first selected pin
set memberIDList [sch::dbGetMemberNets $firstSelectedNetGroupId]
#get the connected instTerms of the first member
set firstMemberId [lindex $memberIDList 0]
set connectIInstTermsList [ sch::dbGetConnectedInstTerms $firstMemberId ]
```

Related Commands

dbGetMemberNets

dbGetConnectedNetName

Returns the name of the net connected to the pin whose in-context object DBID is passed as argument.
This command is applicable for system design objects.

Return Type

STRING

Syntax

```
sch::dbGetConnectedNetName <instTermContextId>
```

Parameters

Parameter	Type	Description
`instTermContextId`	`DBID`	DBID of the pin for which connected net name is being queried This parameter is required.

Examples

```
# This command will return the itemId of the selected item say a system block
set itemId [ sch::dbGetSelectedItems [ sch::dbGetActivePage ]]
puts $itemId
# Output - db:0000001f

# This command will return all the system block pin IDs for the system block
# Only one pin is present in the functional block, so only one ID is displayed.
sch::dbGetBlockPins $itemId
# Output - db:00000018

#Get the pins mapped to this system block pin
sch::dbGetMappedRootDesignPins [ sch::dbGetBlockPins $itemId ]
Output - 0x0000000C 0x0000000D
#command to find the net name connected to the pin.
#the command will return A_POWER which is name of the net connected to this pin
dbGetConnectedNetName 0x0000000C
Output - A_POWER
```

Related Commands

dbGetMappedRootDesignPins

dbGetConnectedNetSpath

dbGetConnectedNetSpath

Returns the spath of the net connected to the pin whose in-context object DBID is passed as argument

Return Type

STRING

Syntax

```
sch::dbGetConnectedNetSpath <instTermContextId>
```

Parameters

Parameter	Type	Description
`instTermContextId`	`DBID`	DBID of the pin for which connected net spath is being queried This parameter is required.

Examples

```
# This command will return the itemId of the selected item say a system block
set itemId [ sch::dbGetSelectedItems [ sch::dbGetActivePage ]]
puts $itemId
# Output - db:0000001f

# This command will return all the system block pin IDs for the system block
# Only one pin is present in the functional block, so only one ID is displayed.
sch::dbGetBlockPins $itemId
# Output - db:00000018

#Get the pins mapped to this system block pin
sch::dbGetMappedRootDesignPins [ sch::dbGetBlockPins $itemId ]
# Output - 0x0000000C 0x0000000D
#command to find the net name connected to the pin.
#the command will return spath of the net connected to this pin
dbGetConnectedNetName 0x0000000C
# Output - @worklib:top(sch_1):\N1
```

Related Commands

dbGetConnectedNetName

dbGetMappedRootDesignPins

dbGetInstanceRefdes

Returns the Reference Designator of the component whose pin's instTermBitContext that is in-context object is passed as a parameter. The component's pin is connected at the system level.

Return Type

STRING

Syntax

```
sch::dbGetInstanceRefdes <instTermContextId>
```

Parameters

Parameter	Type	Description
`instTermContextId`	`DBID`	DBID of the pin in-context object This parameter is required.

Examples

```
# after selecting a NetGroup on the page and running the following sequence of commands
returns the list of netBitContext DBIDs of members of the NetGroup
#get the pageId of the active page
set pgId [sch::dbGetActivePage]
#get the list of dbIDs of all the selected NetGroups on the page
set selectedNetGroupIds [sch::dbGetSelectedItems $pgId]
#get the dbID of the first selected pin in the list of pins' dbIDs
set firstSelectedNetGroupId [lindex $selectedNetGroupIds 0]
#get the data of the first selected pin
set memberIDList [sch::dbGetMemberNets $firstSelectedNetGroupId]
#get the connected instTerms of the first member
set firstMemberId [lindex $memberIDList 0]
set connectedIInstTermsList [ sch::dbGetConnectedInstTerms $firstMemberId ]

#Get the RefDes of the parent instance of that instTerm (pin) using dbGetInstanceRefdes
set firstInstTermID [lindex $connectedIInstTermsList 0]
sch::dbGetInstanceRefdes $firstInstTermID
```

Related Commands

dbGetPinNumber

dbGetConnectedInstTerms

dbGetMemberNets

dbGetMappedBlockName

Returns the name of the block to which this system block is mapped.

Return Type

STRING

Syntax

```
sch::dbGetMappedBlockName <itemId>
```

Parameters

Parameter	Type	Description
`itemId`	`STRING`	The DBID of the selected system block This parameter is required.

Examples

```
#Get the mapped block name of a selected system Block
#First select the system block and then get the selected Items on active Page.
sch::dbGetMappedBlockName [ sch::dbGetSelectedItems [ sch::dbGetActivePage ] ]
```

dbGetMappedBlockNetName

For an instTermBitContext of a pin that is connected at the system level, it returns the name of the net connected to this pin in the block's design.
instTermBitContext is in-context object for a pin vector bit.

Return Type

STRING

Syntax

```
sch::dbGetMappedBlockNetName <instTermContextId>
```

Parameters

Parameter	Type	Description
`instTermBitContextId`	`DBID`	The instTermBitContext DBID of an instTerm which is a member of system block pin and connected at the system-level This parameter is required.

Examples

```
# after selecting a NetGroup on the page and running the following sequence of commands
returns the list of netBitContext DBIDs of members of the NetGroup
#get the pageId of the active page
set pgId [sch::dbGetActivePage]
#get the list of dbIDs of all the selected NetGroups on the page
set selectedNetGroupIds [sch::dbGetSelectedItems $pgId]
#get the dbID of the first selected pin in the list of pins' dbIDs
set firstSelectedNetGroupId [lindex $selectedNetGroupIds 0]
#get the data of the first selected pin
set memberIDList [sch::dbGetMemberNets $firstSelectedNetGroupId]
#get the connected instTerms of the first member
set firstMemberId [lindex $memberIDList 0]
set connectIInstTermsList [ sch::dbGetConnectedInstTerms $firstMemberId ]

set firstInstTermID [lindex $connectIInstTermsList 0]
#Get the name of the connected net to the instTerm
sch::dbGetMappedBlockNetName $firstInstTermID
```

Related Commands

dbGetConnectedInstTerms

dbGetMappedBlockNetSpath

For an instTermBitContext of a pin that is connected at the system level, it returns the SPath of the net connected to this pin in the block's design. instTermBitContext is in-context object for a pin vector bit.

Return Type

STRING

Syntax

```
sch::dbGetMappedBlockNetSpath <instTermContextId>
```

Parameters

Parameter	Type	Description
`instTermContextId`	`DBID`	The instTermBitContext of a pin which is a member of system block pin and connected at the system-level This parameter is required.

Examples

```
# after selecting a NetGroup on the page and running the following sequence of commands
returns the list of netBitContext DBIDs of members of the NetGroup
#get the pageId of the active page
set pgId [sch::dbGetActivePage]
#get the list of dbIDs of all the selected NetGroups on the page
set selectedNetGroupIds [sch::dbGetSelectedItems $pgId]
#get the dbID of the first selected pin in the list of pins' dbIDs
set firstSelectedNetGroupId [lindex $selectedNetGroupIds 0]
#get the data of the first selected pin
set memberIDList [sch::dbGetMemberNets $firstSelectedNetGroupId]
#get the connected instTerms of the first member
set firstMemberId [lindex $memberIDList 0]
set connectIInstTermsList [ sch::dbGetConnectedInstTerms $firstMemberId ]

set firstInstTermID [lindex $connectIInstTermsList 0]
#Get the spath of the connected net to the instTerm
sch::dbGetMappedBlockNetSpath $firstInstTermID
```

Related Commands

dbGetMappedBlockNetName

dbGetMappedBlockType

Returns the type of mapping associated with the system block or pin. The output can be one of the following:

HB_COMPONENTS - The BlockPins are mapped to connector pins of the source block or design.
HB_PORTS_PRESENT - The BlockPins are mapped to the interface nets or pins of the source block or design.
HB_PORTS_NEW - The BlockPins are mapped to new interface ports that do not exist yet. This is the case where the system block is not mapped to any existing design.
BOARD_COMPONENTS - The BlockPins are mapped to components from a board.

Return Type

STRING

Syntax

```
sch::dbGetMappedBlockType <itemId>
```

Parameters

Parameter	Type	Description
`itemId`	`DBID`	The DBID of system block or system block pin This parameter is required.

Examples

```
#Get the mapped block Type by passing system block DBID. Get the DBID of the selected
system block using dbGetSelectedItems on active page
sch::dbGetMappedBlockType [ sch::dbGetSelectedItems [ sch::dbGetActivePage ] ]
```

Related Commands

dbGetMappedBlockName

dbGetMappedBoardItem

Returns the DBID of an in-memory database created for a hierarchical block or board.
An in-memory database of the block or board is loaded and a reference to it is returned in the form of DBID.
This DBID can be used to get further data from the block or board.
The input string can be block name or board name (*.brd or *.mdd)

Return Type

DBID

Syntax

```
sch::dbGetMappedBoardItem<boardName>
```

Parameters

Parameter	Type	Description
`<block or Board Name>`	`STRING`	The name of hierarchical block or board This parameter is required.

Examples

```
#To get the DBID for a board named 'pcb.brd'
sch::dbGetMappedBoardItem "pcb.brd"
Output - db:0000001d

#To get the DBID for a hierarchical block named 'mid'
sch::dbGetMappedBoardItem "mid"
Output - db:0000001c
```

Related Commands

dbGetConnectedBoardPins

dbGetMappedRootDesignPins

This command is used for system-level design projects.

Returns the list of DBIDs of instTermBitContexts which are mapped to a system block pin or to pins of system block.
instTermBitContext is in-context object for a pin vector bit.

Return Type

LIST

Syntax

```
sch::dbGetMappedRootDesignPins <itemId>
```

Parameters

Parameter	Type	Description
`<itemId>`	`DBID`	The DBID of the system block or system block pin This parameter is required.

Examples

```
#Select either a System block or system block pin and use below command to get the list
of mapped instTermBitContexts
sch::dbGetMappedRootDesignPins [ sch::dbGetSelectedItems [ sch::dbGetActivePage ] ]
```

Related Commands

dbGetMappedRootDesignPinSpaths

dbGetMappedRootDesignPinSpaths

This command is used for system-level design projects.
Returns the list of SPaths of pins which are mapped to a system block pin or to pins of system block.

Return Type

LIST

Syntax

```
sch::dbGetMappedRootDesignPinSpaths <itemId>
```

Parameters

Parameter	Type	Description
`itemId`	`DBID`	This is the DBID of either system block or system block pin This parameter is required.

Examples

```
#Select either a System block or system block pin and use below command to get the list
of mapped instTermBitContexts
sch::dbGetMappedRootDesignPinSpaths [ sch::dbGetSelectedItems [ sch::dbGetActivePage ]
]
```

Related Commands

dbGetMappedRootDesignPins

dbGetMemberNets

Returns the list of netBitContext of members of specified NetGroup where netBitContext is in-context object for a net bit.

Return Type

LIST

Syntax

```
sch::dbGetMemberNets <netGroup DBID>
```

Parameters

Parameter	Type	Description
`netGroup DBID`	`DBID`	The DBID of the selected NetGroup This parameter is required.

Examples

```
# Select a NetGroup on the page and run the following commands to get the list of
netBitContext DBIDs of members of the NetGroup
#get the pageId of the active page
set pgId [sch::dbGetActivePage]
#get the list of dbIDs of all the selected NetGroups on the page
set selectedNetGroupIds [sch::dbGetSelectedItems $pgId]
#get the dbID of the first selected NetGroup in the list of NetGroups' dbIDs
set firstSelectedNetGroupId [lindex $selectedNetGroupIds 0]
#get the data of the first selected Netgroup
set memberList [sch::dbGetMemberNets $firstSelectedNetGroupId]
```

dbGetMemberNetSpaths

This command lists the sPaths of all the members of the selected NetGroup.

Return Type

LIST

Syntax

```
sch::dbGetMemberNetSpaths <dbID>
```

Parameters

Parameter	Type	Description
`<dbID>`	`DBID`	dbID of the NetGroup currently selected on the canvas This parameter is required.

Examples

```
#command to store the dbID of the selected NetGroup in a variable called netGrpDbID
#get the sPath of the members of the selected NetGroup
set netGrpDbID [sch::dbGetSelectedItems [sch::dbGetActivePage]]
sch::dbGetMemberNetSpaths $netGrpDbID
#For example: @worklib.mid(tbl_1):\\N3\\ @worklib.mid(tbl_1):\\N4\\
#@worklib.mid(tbl_1):\\N2\\
```

Related Commands

dbGetMemberNetNames

dbGetPinNumber

This command is used for system-level design projects.

Returns the number of the pin whose instTermBitContext is passed as a parameter. The pin is connected at the system level.
instTermBitContext is in-context object for a pin vector bit.

Return Type

INT

Syntax

```
sch::dbGetPinNumber <instTermContextId>
```

Parameters

Parameter	Type	Description
`instTermContextId`	`DBID`	DBID of the instTermBitContext This parameter is required.

Examples

```
# after selecting a NetGroup on the page and running the following sequence of commands
returns the list of netBitContext DBIDs of members of the # NetGroup
# get the pageId of the active page
set pgId [sch::dbGetActivePage]
# get the list of dbIDs of all the selected NetGroups on the page
set selectedNetGroupIds [sch::dbGetSelectedItems $pgId]
# get the dbID of the first selected pin in the list of pins' dbIDs
set firstSelectedNetGroupId [lindex $selectedNetGroupIds 0]
# get the data of the first selected pin
set memberIDList [sch::dbGetMemberNets $firstSelectedNetGroupId]
# get the connected instTerms of the first member
set firstMemberId [lindex $memberIDList 0]
set connectedIInstTermsList [ sch::dbGetConnectedInstTerms $firstMemberId ]

# Get the RefDes of the parent instance of that instTerm (pin) using
dbGetInstanceRefdes
set firstInstTermID [lindex $connectedIInstTermsList 0]
sch::dbGetPinNumber $firstInstTermID
```

Related Commands

dbGetInstanceRefdes

dbGetConnectedInstTerms

dbGetMemberNets

dbIsMappedBlock

Returns 1 if the specified DBID is of a system block or system block pin.

Return Type

DBBOOL

Syntax

```
sch::dbIsMappedBlock <itemId>
```

Parameters

Parameter	Type	Description
`itemId`	`DBID`	The DBID of an item This parameter is required.

Examples

```
#Select an item on canvas and get its DBID and use dbIsMappedBlock API to check if it
is a system block
sch::dbIsMappedBlock [ sch::dbGetSelectedItems [ sch::dbGetActivePage ] ]
#Output = 1
```

dbIsPinConnectedToPowerNetInMappedBlock

It checks if the pin whose instTermBitContext is passed as input is connected to the power net. It returns 1 if the pin is connected and 0 if the pin is not connected. Also, it returns 0 if the bit context of the given item is NULL.
It checks the connection to power net for the pin in the design which is mapped to system block.
instTermBitContext is in-context object for a pin vector bit.

Return Type

DBBOOL

Syntax

```
sch::dbIsPinConnectedToPowerNetInMappedBlock <itemId>
```

Parameters

Parameter	Type	Description
`<instTermBitContext>`	`DBID`	DBID of a instTermBitContext This parameter is required.

Examples

```
# This command will return the itemId of the selected item say a system block
set itemId [ sch::dbGetSelectedItems [ sch::dbGetActivePage ]]
puts $itemId
# Output - db:0000001f

# This command will return all the system block pin IDs for the system block
# Only one pin is present in the functional block, so only one ID is displayed.
sch::dbGetBlockPins $itemId
# Output - db:00000018

#Get the pins mapped to this system block pin
sch::dbGetMappedRootDesignPins [ sch::dbGetBlockPins $itemId ]
Output - 0x0000000C 0x0000000D

#command to find if the pin is connected to a power net
sch::dbIsPinConnectedToPowerNetInMappedBlock db:0000000C
```

Related Commands

dbGetMappedRootDesignPins

dbGetConnectedInstTerms

10

Team Design

checkin

Check in the edited design object(s) to the shared area

Return Type

STRING

Syntax

```
cdsdm -proj <project path> checkin -object <object name> -type <major/minor> ?[-comment
<comment description>]? ?[-keep_checkedout <true/false>]?
```

Parameters

Parameter	Type	Description
`-proj <project path>`	`STRING`	Path to the project file This parameter is required.
`-object <object name>`	`LIST OF STRING`	Name of the design object(s) to be checked in This parameter is required.
`-type <major/minor>`	`STRING`	Check-in type for the design object(s) to be checked in This parameter is required.
`-comment <comment description>`	`STRING`	Comment for the design object(s) to be checked in This parameter is optional.
`-keep_checkedout <true/false>`	`BOOL`	Checked in design object(s) and kept it in checkout state This parameter is optional.

checkout

Check out design object(s) for editing

Return Type

STRING

Syntax

```
cdsdm -proj <project path> checkout -object <object name> ?[-version <version name>]?
```

Parameters

Parameter	Type	Description
`-proj <project path>`	`STRING`	Path to the project file This parameter is required.
`-object <object name>`	`LIST OF STRING`	Name of the object(s) to be checked out This parameter is required.
`-version <version name>`	`STRING`	Version of the design object(s) to be checked out This parameter is optional.

downloadAttachment

Downloads the attachment corresponding to CDS_LIBRARY_ID to the folder specified in the input json file.

Return Type

NONE

Syntax

```
cpunicorn::downloadAttachment <json>
```

Examples

```
cpunicorn::downloadAttachment \{"search":\{"CDS_LIBRARY_ID":"dummy_id"\},"folders":\
{".*":"[cps::getProjectMainDir]"\}\}
# json dummy_id is the id of CDS LIBRARY.
#folders key is the extension of the file to be downloaded and Directory is where it is
downloaded
```

enableteamdesign

Enable a project for team design

Return Type

STRING

Syntax

```
cdsdm -proj <project path> enableteamdesign -vault <vault location> -product
<workspace> -project <project name>
```

Parameters

Parameter	Type	Description
`-proj <project path>`	`STRING`	Path to the project file This parameter is required.
`-vault <vault location>`	`LIST OF STRING`	Location of the vault where data is stored This parameter is required.
`-product <workspace>`	`STRING`	Workspace of object This parameter is required.
`-project <project name>`	`STRING`	Name of the project to be enabled for team design This parameter is required.

Examples

```
cdsdm -proj /servers/scratch05/dheeren/workspace/project/project.cpm enableteamdesign -
vault /servers/scratch05/dheeren/sharedarea -product etdws
```

exportPCBPartition

Exports the subsystems to separate projects.

Return Type

BOOL

Syntax

```
exportPCBPartition [list {-partition <Subsystem_name> -project_name <project_name> -
design_name <design_name> -path <project_path>}]
```

Examples

```
#Command to export a single subsystem named subsystem_1
exportPCBPartition [list {-partition "Subsystem_1" -project_name "subsystem_1" -
design_name "subsystem_1" -path "../subsystem_1"}]

#Command to export 2 subsystems named subsystem_1, subsystem_2
exportPCBPartition [list {-partition "Subsystem_1" -project_name "subsystem_1" -
design_name "subsystem_1" -path "../subsystem_1"} {-partition "Subsystem_2" -
project_name "subsystem_2" -design_name "subsystem_2" -path "../subsystem_2"}]
```

getDesignListURL

To get URL of Pulse design list.

Return Type

const char*

Syntax

```
cpbf:::getDesignListURL
```

getPulseStatus

Returns current state of Pulse.

Return Type

STRING

Syntax

```
cps::getPulseStatus
```

Examples

```
Tcl&gt; cps::getPulseStatus
PULSE_STATE_MAINTENANCE
Tcl&gt; cps::getPulseStatus
PULSE_STATE_RUNNING
```

importCaptureDesign

Return Type

INT

Syntax

```
sch::importCaptureDesign
```

joinproject

Join a project enabled for team design

Return Type

STRING

Syntax

```
cdsdm joinproject -vault <vault location> -product <workspace> -project <project name>
-workarea <work area> [-label <label name>]
```

Parameters

Parameter	Type	Description
`-vault <vault location>`	`STRING`	Location of the vault where data is stored This parameter is required.
`-product <workspace>`	`STRING`	workspace of design This parameter is required.
`-project <project name>`	`STRING`	Name of the project to be joined This parameter is required.
`-workarea <work area>`	`STRING`	Local copy of the project This parameter is required.
`-label <label name>`	`STRING`	Name of the label with which the project is to be joined This parameter is optional.

Examples

```
cdsdm joinproject -vault lnx-vikass:1699 -product depot/etdws -project etdproj -work
/servers/scratch05/dheeren/workArea/jpworkspace
```

openDiagnosticsDialog

Launches the Generate Diagnostics dialog box to generate a medic test case for root cause analysis by Cadence

Return Type

INT

Syntax

```
cpunicorn::openDiagnosticsDialog
```

Examples

```
cpunicorn::openDiagnosticsDialog
```

removelabel

Remove label from the design object(s)

Return Type

STRING

Syntax

```
cdsdm -proj <project path> removelabel -object <object name> -label <label name>
```

Parameters

Parameter	Type	Description
`-proj <project path>`	`STRING`	Path to the project file This parameter is required.
`-object <object name>`	`LIST OF STRING`	Name of the design object(s) to which the label is to be applied This parameter is required.
`-label <label name>`	`STRING`	Name of the label to be removed from the design object(s) This parameter is required.

rollback

Discard changes to the design object(s), revert to its previous state, keep the object(s) checked out

Return Type

STRING

Syntax

```
cdsdm -proj <project path> rollback -object <object name>
```

Parameters

Parameter	Type	Description
`-proj <project path>`	`STRING`	Path to the project file This parameter is required.
`-object <object name>`	`LIST OF STRING`	Name of the design object(s) whose changes are to be rolled back This parameter is required.

shareDesign

Opens share design UI.

Return Type

BOOL

Syntax

```
cpbf::shareDesign
```

Parameters

Parameter	Type	Description
`bReload`	`BOOL`	Reload share UI This parameter is optional. Default value is `false`.

undocheckout

Discard the changes, cancel the checkout, and return the design object(s) to its previous state

Return Type

STRING

Syntax

```
cdsdm -proj <project path> undocheckout -object <object name>
```

Parameters

Parameter	Type	Description
`-proj <project path>`	`STRING`	Path to the project file This parameter is required.
`-object <object name>`	`LIST OF STRING`	Design object(s) whose changes are to be discarded and whose checkout is cancelled This parameter is required.

update

Update the design object(s) to the latest changes in the shared area

Return Type

STRING

Syntax

```
cdsdm -proj <project path> update -object <object name>
```

Parameters

Parameter	Type	Description
`-proj <project path>`	`STRING`	Path to the project file This parameter is required.
`-object <object name>`	`LIST OF STRING`	Name of the design object(s) to be updated This parameter is required.

11

Variant Management

addAlternatePart

Adds an alternate component to a selected existing component in the current variant of the design. Only one component can be selected at a time.
Returns NONE on success and either "incomplete" or "command exits with error" or failure.

Return Type

NONE

Syntax

```
addAlternatePart <lib> <cell> <view> <partname> -n "1" -key [list <key_properties>] -i
[list <injected_properties>]
```

Parameters

Parameter	Type	Description
`lib`	`STRING`	Name of the library in which the component resides. This parameter is required.
`cell`	`STRING`	Name of the cell in the specified library enclosed within straight double quotes (" "). This parameter is required.
`view`	`STRING`	Version number of the component to be added. This parameter is required.
`partname`	`STRING`	Name of the cell to be added in uppercase and enclosed within double quotes (" "). This parameter is required.
`-n "1" -key [list <key_properties>]`	`STRING`	List of key properties along with their values. This parameter is required.
`-i [list <injected_properties>]`	`STRING`	List of injected properties and their values. To syntax for specifying property values is: "property_name = property_value" This parameter is required.

Examples

```
addAlternatePart discrete "cap" sym_1 "CAP" -n "1" -key [list "VOLTAGE=50V"
"MATERIAL=C0G-CERM" "TOLERANCE=+/-0.1PF" "VALUE=6.0PF" "PACK_TYPE=0402" "PART_NAME=CAP"
] -i [list "VOLTAGE=50V" "QUAD_MODEL=CAP_6.0PF" "TOLERANCE=+/-0.1PF" "JEDEC_TYPE=CAP-
0402-HP55" "VALUE=6.0PF" "ALT_SYMBOLS=(CAP-0402-HP55-FLEX-CL-P25-3,CAP-0402-HP55-FLEX-
LPI-P05-4)" "PART_NUMBER=131S0308" ]
```

applyHierVar

This command applies the low block variant information at top design variant.
On success, the variant overrides should be visible on the canvas/varianttable corresponding to the low block variant specified in the command for the given top design variant. In case of multi-select, same block instances should be selected.

Return Type

NONE

Syntax

```
applyHierVar -variant <top_level_variant> -spath [list <block_spaths>] -blockvariant
<block_variant>
```

Parameters

Parameter	Type	Description
`-variant`	`STRING`	Name of the top design variant This parameter is required.
`-spath`	`STRING`	List of spaths of the block instances of a block eligible for hierarchical variants This parameter is required.
`-blockvariant`	`STRING`	Name of low level variant to be applied at the top design variant This parameter is required.

Examples

```
#to apply block variant LOWVAR1 on the selected block instances for the top design
variant TOPVAR11

applyHierVar -variant TOPVAR11 -spath [list @worklib.testapply1(tbl_1):\\I11\\
@worklib.testapply1(tbl_1):\\I8\\ ] -blockvariant LOWVAR1

#to apply block variant Base on the selected block instances for the top design variant
TOPVAR11

applyHierVar -variant TOPVAR11 -spath [list @worklib.testapply1(tbl_1):\\I11\\
@worklib.testapply1(tbl_1):\\I8\\ ] -blockvariant Base
```

createVariant

Creates a variant of the current design.

Return Type

NONE

Syntax

```
createVariant -variant <variant_name>
```

Parameters

Parameter	Type	Description
`-variant <variant_name>`	`STRING`	name of the variant This parameter is required.

Examples

```
#Command to create a variant of the base design named as "JAPAN"
createVariant -variant JAPAN
```

deleteVariant

Deletes the specified variant of the design from the project.

Return Type

NONE

Syntax

```
deleteVariant <name_of_variant>
```

Parameters

Parameter	Type	Description
`<name_of_variant>`	`STRING`	name of the variant to be deleted This parameter is required.

Examples

```
#Command to delete the variant "INDIA" from the project
deleteVariant INDIA
```

disableHierVar

This command unloads the variant data of the low block from the top design variant database. All block variant information is removed from the top design and the block comes back to its original variant state at top level. On success, all block variant information will be removed from all top level variants.

Return Type

NONE

Syntax

```
disableHierVar -block <block_name>
```

Parameters

Parameter	Type	Description
`-block`	`STRING`	Name of the low block whose variant information needs to be removed from top design This parameter is required.

Examples

```
#to unload LOW1 block hv information from the top design

disableHierVar -block LOW1
```

editVariant

This command edits the variant of the design. The name of the variant or the custom variables or both can be modified using this command.

Return Type

INT

Syntax

```
editVariant -variant <variant_name> ?[-custom [list {<custom_var_name>=<new_value>}]]?
```

Parameters

Parameter	Type	Description
`<variant_name>`	`STIRING`	Name of the variant. This parameter is required.
`[-custom [list {<custom_var_name>=<new_value>}]]`	`STRING`	Custom variable name and the new value of the variable. Custom variable and its new value must be specified within curly braces {}. For example, -custom [list {AZTEC=ny}] This parameter is optional.

Examples

```
editVariant -variant JAPAN
```

exportVariantLst

Creates a variants.lst file under physical folder which is used by Allegro.
This file contains the information about the different variants in the design and the parts added to those variants.

Return Type

NONE

Syntax

```
exportVariantLst
```

Examples

```
exportVariantLst
```

getActiveVariant

Returns the currently set variant in variant canvas mode in System Capture
Returns empty in case of no variant set

Return Type

variant name

Syntax

```
getActiveVariant
```

Examples

```
-&gt;getActiveVariant
-&gt;INDIA
```

Related Commands

viewVariant

loadVariantDatabase

To load the variant database in System Capture. The variant.dat file from the variant view is used for this.

Return Type

NONE

Syntax

```
cpb::loadVariantDatabase
```

Examples

```
cpb::loadVariantDatabase
```

refreshHierVar

This command updates the low level block(s) variant information at top design variant database. If some variant information is modified at low block level, then on refresh this information will be updated or synced at top design variant database.

Return Type

NONE

Syntax

```
refreshHierVar -all |-block <block_name>
```

Parameters

Parameter	Type	Description
`-all`	`NONE`	For updating variant information of all hv blocks This parameter is required.
`-block`	`STRING`	Name of the low level block whose variant information needs to be updated at top design This parameter is required.

Examples

```
# to update LOW1 block variant information at top design variant database
refreshHierVar -block LOW1

# to update variant information of all hv eligible blocks at top design
refreshHierVar -all
```

resetVariant

Resets the component to its initial state for the current variant.

Return Type

INT

Syntax

```
resetVariant -variant <variant_name> -comp <component_name>
```

Parameters

Parameter	Type	Description
`-variant <variant_name>`	`STRING`	Name of the variant. This parameter is required.
`-comp <component_name>`	`STRING`	Name of the component. This parameter is required.

Examples

```
resetVariant -variant QWERTY -comp U2
```

saveVariantDB

saves the variant DB

Return Type

NONE

Syntax

```
saveVariantDB
```

Examples

```
saveVariantDB
```

setAlternate

It changes the priority of the alternate part of the variant.

Return Type

INT

Syntax

```
setAlternate -variant <variant_name> -comp <component_name> -oldIndex <value> -newIndex
<value>
```

Parameters

Parameter	Type	Description
`-variant <variant_name>`	`STRING`	Name of the variant. This parameter is required.
`-comp <component_name>`	`STRING`	Component name. This parameter is required.
`-oldIndex <value>`	`INT`	Initial priority. This parameter is required.
`-newIndex <value>`	`INT`	Final priority. This parameter is required.

Examples

```
setAlternate -variant USA -comp U1 -oldIndex 1 -newIndex 2
```

setDNI

It sets or removes the Do Not Install (DNI) attribute of the selected object of the specified variant. No specific variant needs to be open for this command to work.

Return Type

NONE

Syntax

```
setDNI -variant <variant_name> -comp <comp_name> -state <choice>
```

Parameters

Parameter	Type	Description
`-variant <variant_name>`	`STRING`	The variant name for which the object attribute has to be set as Do Not Install. This must be specified in uppercase. This parameter is required.
`-comp <comp_name>`	`STRING`	The name of the object whose attribute has to be set as Do Not Install. This parameter is required.
`-state <choice>`	`BOOLEAN`	Option to install or to not install the object in a variant. This parameter is required.

Examples

```
#command to set an object C100 in variant ASDF as DNI
setDNI -variant ASDF -comp C100 -state true
#If the object C100 was already DNI, no changes
#will be made. If it was not DNI, it will become DNI.
#command to set an object C100 in variant ASDF as non-DNI
setDNI -variant ASDF -comp C100 -state false
#If the object C100 was already non-DNI, no changes
#will be made. If it was DNI, it will become non-DNI.
```

variantEditor

Used to control the Variant Spreadsheet mode. There are various switches that can be used with this command and vary according to the active tab.

For Variant Summary Tab -
1) variantEditor -create -variant <newvariantname>
Used to create a new variant while in spreadsheet mode.

2) variantEditor -edit -variant <existingvariantname> -newName <newvariantname>
Used to edit an existing variant while in spreadsheet mode.

3) variantEditor -selectRow {refdes}
Used to highlight or select a row in the variant editor.

4) variantEditor -delete <variantnames>
Used to delete or remove the variant or a list of variants from the database.

5) variantEditor -copy -variant <existingvariantname>
Used to create a variant which is a copy of an existing variant.

6) variantEditor -removeAlt -variant <existingvariantname> -comp <refdes> -index <value>
Used to remove an alternate of the component in a variant with refdes <refdes> and index as <value>.

7) variantEditor -selectCell -funcName <funcgrpname> -variant -comps {selection-range}

Used to select a cell in the Variant Spreadsheet from one of the refdes base or variant columns. There can be multiple values defined for multiple selections of cells.

8) variantEditor -selectCell -variant -selectCol <columnnumber>
Used to select one column as a whole

9) variantEditor clearSelection
Used to clear the cell selected in Variant Editor

10) variantEditor -dni <single component or range of components> <dnistate>
Used to mark a single component or range of components DNI in Variant Editor depending on the existing dni state. It toggles the previous state.
This command is available in Function and Alternate Tabs also.

11) variantEditor -reset <single component or range of components>

Used to set a single or multiple components back to base schematic value
This command is available in Function and Alternate Tabs also.

12) variantEditor -includeFuncGrpToVariant -fg <funcgrpname> -var <variantname>
Used to include a function group to a variant in Variant Editor

13) variantEditor -excludeFuncGrpFromVariant -fg <funcgrpname> -var <variantname>
Used to exclude a function group from a variant in Variant Editor

14) variantEditor -setActiveTab <tabname>
Used to set an active tab out of the four tabs in Variant Editor

For Variant Details Tab -
1) variantEditor -varDet -var {variantname}
Used to set the variant in the combo box of the Variant details tab for which details can be viewed (like included function group/individual components)

2) variantEditor -selectCell -varDetailTab -funcName {functionname} -comps { selection-range }
Used to select cells in the Variant Detail tab

For Alternates Tab -
1) variantEditor -selectCell -altTab -altbaseTable -comps { selection-range }
Used to select cells in the Alternates tab

2) variantEditor -addToAltCompList -refdes { selection-range}
Used to add refdes to the Alternate Component List

3) variantEditor -removeFromAltCompList -refdes { selection-range}
Used to remove refdes(s) from the Alternate Component List.

4) variantEditor -removeAlt -alt {} -comp {<refdes>} -status {<status_in_Alt>}
Used to remove an alternate part row for a given refdes

For Function Group Tab -
1) variantEditor -create -function {functionname} -desc {description}
Used to create a new function group with a given description.

2) variantEditor -edit -function {oldfunctionname} -newName {newfunctionname} -desc {description}
Used to edit the name or description of an existing function group.

3) variantEditor -copy -function {functionname}
Used to create a copy an existing function group.

4) variantEditor -delFuncGrp {functionname}
Used to delete an existing function group.

5) variantEditor -selectCell -funcTab -funcTable -funcName {functionname} -comps { selection-range }
Used to select a function name or component(s) in a function group

6) variantEditor -addToFuncGroup -refdes { selection-range } -fg {functionname}
Used to component(s) to a function group

7)variantEditor -removeFromFuncGroup -refdes { selection-range } -fg {functionname}
Used to remove component(s) from a function group

8) variantEditor -makeAlternate {<refdes>} {Pref} {<status_in_Alt>}
Used to make a Preferred part row into a specified alternate part row for a given refdes.
This command is also available in Alternate Tab

9) variantEditor -makePreferred {<refdes>} {<status_in_Alt>}
Used to make an Alternate part row into a Preferred part row for a given refdes.
This command is also available in Alternate Tab

10) variantEditor -removeAlt -func {functionname} -comp {<refdes>} -status {<status_in_Alt>}
Used to remove an alternate part row for a given refdes and given function group

11) variantEditor -refreshFromAlternate -fg {functionname} -refdes { selection-range}
Used to restore the refdes(s) to the version in Alternate Tab. All Function/Variant overrides are removed.
Available both in Function and Variant Summary Tabs.

For Addition and Deletion of Property Columns from UI -

variantEditor -addPropCol { space separated property names}
variantEditor -removePropCol { space separated property names}

Return Type

NONE

Syntax

```
variantEditor -<switches>
```

Parameters

Parameter	Type	Description
`-create -variant <newvariantname>`	STRING	Name of the new variant to be created This parameter is required.
`-edit -variant <existingvariantname> -newName <newvariantname>`	STRING	Name of an existing variant which needs to be edited, and the new name of the variant This parameter is required.
`-selectRow {refdes}`	STRING	Refdes of the component to be selected in UI This parameter is required.
`-delete <variantnames>`	STRING	Name of an existing variant to be deleted This parameter is required.
`-copy -variant <existingvariantname>`	STRING	Name of an existing variant to be copied This parameter is required.
`-removeAlt -variant <existingvariantname> -comp <refdes> -index <value>`	STRING	Name of an existing variant, refdes under that variant and index of the refdes alternate which needs to be removed This parameter is required.
`-selectCell -funcName <funcgrpname> -variant -comps {selection-range}`	STRING	Name of the function group, variant and the range of components which need to be selected This parameter is required.
`-selectCell -variant -selectCol <columnnumber>`	INT	Column number to be selected as a whole This parameter is required.

`-clearSelection`	none	Switch to clear the selected row/columns in the table This parameter is required.
`-dni <single component or range of components> <dnistate>`	STRING	Refdes or list of Refdeses to be marked as DNI, new DNI state This parameter is required.
`-reset <single component or range of components>`	STRING	Refdes or list of Refdeses to be set back to base schematic values This parameter is required.
`-includeFuncGrpToVariant -fg <funcgrpname> -var <variantname>`	STRING	Name of the function group which needs to be included to the variant, name of the variant This parameter is required.
`-setActiveTab <tabname>`	STRING	Name of the tab which needs to be set in UI out of the 4 tabs This parameter is required.
`-varDet -var {variantname}`	STRING	Name of the variant whose details need to be seen This parameter is required.
`-selectCell -varDetailTab -funcName {functionname} -comps { selection-range }`	STRING	Name of the function group and range of components to be selected This parameter is required.
`-selectCell -altTab -altbaseTable -comps { selection-range }`	STRING	Range of components to be selected This parameter is required.
`-addToAltCompList -refdes { selection-range}`	STRING	Range of components to be added to the alternate component list This parameter is required.

`-removeFromAltCompList -refdes { selection-range}`	`STRING`	Range of components to be removed from the alternate component list This parameter is required.
`-removeAlt -alt {} -comp {<refdes>} -status {<status_in_Alt>}`	`STRING`	Refdes and index of the refdes alternate which needs to be removed from Alternate tab This parameter is required.
`-create -function {functionname} -desc {description}`	`STRING`	Name of the new function group to be created and its description This parameter is required.
`-edit -function {oldfunctionname} -newName {newfunctionname} -desc {description}`	`STRING`	Name of existing function group, new name and description (new/old) This parameter is required.
`-copy -function {functionname}`	`STRING`	Name of an existing function group which needs to be copied This parameter is required.
`-delFuncGrp {functionname}`	`STRING`	Name of an existing function group which needs to be deleted This parameter is required.
`-selectCell -funcTab -funcTable -funcName {functionname} -comps { selection-range }`	`STRING`	Name of the function group, range of components to be selected under that function group This parameter is required.
`-addToFuncGroup -refdes { selection-range } -fg {functionname}`	`STRING`	Range of components to be added to the given function name This parameter is required.
`-removeFromFuncGroup -refdes { selection-range } -fg {functionname}`	`STRING`	Range of components to be deleted from the given function name This parameter is required.

`-makeAlternate {<refdes>} {Pref} {<status_in_Alt>}`	STRING	Refdes which needs to be made as alternate row, status of desired alternate This parameter is required.
`-makePreferred {<refdes>} {<status_in_Alt>}`	STRING	Refdes which needs to be made Preferred, status of source alternate This parameter is required.
`-removeAlt -func {functionname} -comp {<refdes>} -status {<status_in_Alt>}`	STRING	Name of an existing function group, refdes under that function and index of the refdes alternate which needs to be removed This parameter is required.
`-refreshFromAlternate -fg {functionname} -refdes { selection-range}`	STRING	Range of compnents which need to be refreshed from Alternate for the given function name This parameter is required.
`-addPropCol { space separated property names}`	STRING	List of property name columns to be displayed in UI This parameter is required.
`-removePropCol { space separated property names}`	STRING	List of property name columns to be removed from UI This parameter is required.

Examples

```
For Variant Summary Tab -
variantEditor -create -variant XYZ
variantEditor -edit -variant XYZ
variantEditor -selectRow {C1}
variantEditor -delete XYZ
variantEditor -copy -variant XYZ
variantEditor -removeAlt -variant XYZ -comp C1 -index 1
variantEditor -selectCell -funcName {Components} -variant -comps {
Components,C1,TOP_VAR11}
variantEditor -selectCell -variant -selectCol {5}
variantEditor -dni { Components,C1,TOP_VAR11::Components,C4,TOP_VAR11 } 1
variantEditor -reset { Components,C1,TOP_VAR11::Components,C4,TOP_VAR11}
variantEditor -includeFuncGrpToVariant -fg {COPY-FF11} -var {TOP_VAR11}
variantEditor -excludeFuncGrpFromVariant -fg {FF1111} -var {COPY-TOP_VAR11}
variantEditor -setActiveTab {VAR_DET}

variantEditor -varDet -var {INDIA}
variantEditor -selectCell -varDetailTab -funcName {Components} -comps {
Components,C3,RefDes,DNI }


variantEditor -selectCell -altTab -altbaseTable -comps { C3,RefDes,Base::C6,RefDes,Base
}
variantEditor -addToAltCompList -refdes { C3,RefDes,Base::C6,RefDes,Base}
variantEditor -removeFromAltCompList -refdes { C2,RefDes,Pref::C3,RefDes,Pref}
variantEditor -removeAlt -alt {} -comp {C2} -status {Alt1}

variantEditor -create -function {FUNC1} -desc {this is a sample.}
variantEditor -edit -function {FUNC1} -newName {FUNC14576} -desc {this is a sample.}
variantEditor -copy -function {FUNC14576}
variantEditor -delFuncGrp {COPY-FUNC14576}
variantEditor -selectCell -funcTab -funcTable -funcName {WWWSD} -comps {
WWWSD,,FunctionGroups, } #select function group
variantEditor -selectCell -funcTab -funcTable -funcName {WWWSD} -comps {
WWWSD,C2,RefDes,Alt1 } #select component in a function group
variantEditor -addToFuncGroup -refdes { C1,RefDes,Base::C2,RefDes,Base } -fg {SSSS}
variantEditor -removeFromFuncGroup -refdes { SSSS,C1,RefDes,Alt1} -fg {SSSS}
variantEditor -makeAlternate {C2} {Pref} {Alt2}
variantEditor -makePreferred {C2} {Alt1}
variantEditor -removeAlt -func {SSSS} -comp {C2} -status {Alt1}

variantEditor -refreshFromAlternate -fg {SSSS} -refdes { SSSS,C2,RefDes,Alt1}

variantEditor -addPropCol { LOCATION VALUE VOLTAGE}
variantEditor -removePropCol { LOCATION VALUE}
```

variantOFF

Sets the variant mode to off. Sets the tool back to base view.

Return Type

NONE

Syntax

```
variantOFF
```

Examples

```
variantOFF
```

Related Commands

viewVariant

variantSetup

Allocates values to different variant attributes.

Return Type

NONE

Syntax

```
variantSetup [-var_name <value>] [-dni_name <value>] [-dni_value <value>] [-props {
<value> <value> }] [-font_face {<value>}] [-font_size <value>] [-font_color <value>] [-
font_opacity <value>] [-underline <value>] [-bold <value>] [-italics <value>] [-
line_width <value>] [-line_style <value>] [-line_color <value>] [-fill_color <value>]
[-inst_opacity <value>] [-show_cross <value>] [-cross_color <value>]
```

Parameters

Parameter	Type	Description
`[-var_name <value>]`	`STRING`	Variant property name. This parameter is optional.
`[-dni_name <value>]`	`STRING`	Variant value. This parameter is optional.
`[-dni_value <value>]`	`STRING`	DNI value. This parameter is optional.
`[-props { <value> <value> }]`	`STRING`	Displayed column names. This parameter is optional.
`[-font_face {<value>}]`	`STRING`	Font type of the instance property. This parameter is optional.

`[-font_size <value>]`	`INT`	Font size of the instance property. This parameter is optional.
`[-font_color <value>]`	`STRING`	Font color of the instance property. This value is specified in hex color code. This parameter is optional.
`[-font_opacity <value>]`	`INTEGER`	Transparency of the instance property. This parameter is optional.
`[-underline <value>]`	`INTEGER`	Underline option for the instance property. This parameter is optional.
`[-bold <value>]`	`INTEGER`	Bold option for the instance property. This parameter is optional.
`[-italics <value>]`	`INTEGER`	Italics option for the instance property. This parameter is optional.
`[-line_width <value>]`	`INTEGER`	Line width of the variant component. This parameter is optional.
`[-line_style <value>]`	`STRING`	Line style of the instance. This parameter is optional.
`[-line_color <value>]`	`STRING`	Line color of the instance. This value is specified in hex color code. This parameter is optional.
`[-fill_color <value>]`	`STRING`	Variant component color. This value is specified in hex color code. This parameter is optional.
`[-inst_opacity <value>]`	`INTEGER`	Transparency of the component instance. This parameter is optional.
`[-show_cross <value>]`	`INTEGER`	Show cross option on DNI. This parameter is optional.

`[-cross_color <value>]`	`STRING`	Cross color. This value is specified in hex color code. This parameter is optional.

Examples

```
variantSetup -var_name VARIANT_aus -dni_name aus -dni_value DNI_aus -props { REFDES} -
font_face {Arial Unicode MS} -font_size 10 -font_color #769f20 -font_opacity 10 -bold 0
-italics 0 -underline 0 -line_width 10 -line_style dash-dot -line_color #769f20 -
fill_color #769f20 -inst_opacity 10 -show_cross 0 -cross_color #769f20
```

viewVariant

Puts the design in the Variant mode.

Return Type

NONE

Syntax

```
viewVariant -variant <variant_name>
```

Parameters

Parameter	Type	Description
`-variant`	`STRING`	Variant name to be viewed This parameter is required.

Examples

```
#Command to view the variant "GAMMA" schematic
viewVariant -variant GAMMA
```

Related Commands

variantOff

12

Workflow

getConfiguration

Returns the global configuration for all projects on the Pulse server

Return Type

STRING

Syntax

```
cpfm::getConfiguration
```

Examples

```
#To get the configuration when running System Capture:
cpfm::getConfiguration

#The expected output of this command is the saved configuration.
```

Related Commands

cpfm::setPulseConfigValue

setPulseConfigValue

Provides the configuration path and value as parameters for all projects on the Pulse server in the multi-user environment.

Return Type

INT

Syntax

```
cpfm::setPulseConfigValue <confpath> <confvalue>
```

Parameters

Parameter	Type	Description
`confpath`	`String`	Specifies a JSONPath to the required element in the search.config file This parameter is required.
`confvalue`	`String`	Specifies the value of the element in confpath This parameter is required.

Examples

```
#The Tcl command to update the "workflow" subsection under the "unicorn" section in the
search.config file is as follows:

cpfm::setPulseConfigValue unicorn.workflow \[\{"tools":\[\
{"name":"flowmanager","value":"CopyOfCommonTool"\}\],"name":"generic_workflow","object_
type":""\}\]

#Expected return is an integer value (1 for success and 0 for failure of execution of
command): 1
#Note: Refer to the search.config file for the configuration path parameter.
```

Related Commands

cpfm::getConfiguration

13

Workspace

addActionToContextMenu

Adds a menu item to a context menu. The context menu type is returned by the getContextMenuType command. The separator is added before the menu item specified in the beforeMenuItem parameter. On success, the ID of the created action item is returned. If the command fails, the failure reason is returned as a string. Running the command on a previously associated menu item returns the ID of the associated action.

Return Type

INT

Syntax

```
addActionToContextMenu <contextMenuType> <displayLabel> <command> <iconFile> <toolTip>
<shortcut> <context>
```

Parameters

Parameter	Type	Description
`contextMenuType`	`INT`	Type of the context menu returned by getContextMenuType function This parameter is required.
`displayLabel`	`STRING`	Label of the menu item This parameter is required.
`command`	`STRING`	Function to be called when the menu item is clicked This parameter is required.
`iconFile`	`STRING`	The path to the icon file. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32 and 48x48. This parameter is required.
`tooltip`	`STRING`	Tooltip text. This is currently unused This parameter is required.
`shortcut`	`STRING`	Shortcut sequence. This is currently unused This parameter is required.
`context`	`STRING`	The context in which this action is supported. For schematic canvas actions, this should be "sch" This parameter is required.

Examples

```
#in this snippet, a menu item is added to the context menu that is shown
#when a single wire is selected.
#when the menu item is clicked, a message box displaying the name of
#the wire is shown.
set wireContextMenuType [getContextMenuType "sch" {Wire} 0]
set wireInfoCommand {cps::showMessageDialog "Signal Info" "Signal Name -
[sch::dbGetItemName [sch::dbGetSelectedItems [sch::dbGetActivePage]]]" 2 "OK" {}}
set infoIcon [cps::getResourceFullPath {common/themes/light/24x24/info.png}]
set menuLabel "Wire - More Info"
#command to add the action to the context menu
addActionToContextMenu $wireContextMenuType $menuLabel $wireInfoCommand $infoIcon {} {}
sch
#command to add a separator before the newly added action
addSeparatorToContextMenu $wireContextMenuType $menuLabel
```

Related Commands

addActionToContextMenuEx

addSeparatorToContextMenu

getContextMenuType

getResourceFullPath

addActionToContextMenuEx

It is the extended version of the addActionToContextMenu command. It takes two additional icon parameters representing the selected state and the hover state of the context menu item.

See addActionToContextMenu for more details.

Return Type

INT

Syntax

```
addActionToContextMenuEx <contextMenuType> <displayLabel> <command> <iconFile>
<selectedIcon> <hoverIcon> <toolTip> <shortcut> <context>
```

Parameters

Parameter	Type	Description
`contextMenuType`	`INT`	Type of the context menu returned by getContextMenuType function This parameter is required.
`displayLabel`	`INT`	Label of the menu item This parameter is required.
`command`	`STRING`	Function to be called when the menu item is clicked This parameter is required.
`iconFile`	`STRING`	The path to the icon file. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32 and 48x48. This parameter is required.
`selectedIconFile`	`STRING`	The path to the icon file for the selected state of the item. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32 and 48x48. This parameter is required.
`hoverIconFile`	`STRING`	Icon to show on mouse hover. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32 and 48x48. This parameter is required.
`toolTip`	`STRING`	Tooltip text. This is currently unused This parameter is required.
`shortcut`	`STRING`	Shortcut sequence. This is currently unused This parameter is required.
`context`	`STRING`	The context in which this action is supported. For schematic canvas actions, this should be "sch" This parameter is required.

Examples

```
#in this snippet, an action is added to the context menu for single wire segments.
#a message box displaying the wire name is shown when this menu is clicked.
set wireContextMenuType [getContextMenuType "sch" {Wire} 0]
set wireInfoCommand {cps::showMessageDialog "Signal Info" "Signal Name -
[sch::dbGetItemName [sch::dbGetSelectedItems [sch::dbGetActivePage]]]" 2 "OK" {}}
set infoIcon [cps::getResourceFullPath {common/themes/light/24x24/info.png}]
set hoverIcon [cps::getResourceFullPath {common/themes/light/24x24/info.png}]
set selectedIcon [cps::getResourceFullPath {common/themes/light/24x24/info.png}]
set menuLabel "Wire - More Info"
#command to add the action to the context menu
addActionToContextMenuEx $wireContextMenuType $menuLabel $wireInfoCommand $infoIcon
$selectedIcon $hoverIcon {} {} sch
#command to add a separator before the newly added action
addSeparatorToContextMenu $wireContextMenuType $menuLabel
```

Related Commands

addActionToContextMenu

addSeparatorToContextMenu

getContextMenuType

getResourceFullPath

addActionToMenu

Adds a menu in the parent menu. Clicking the menu executes the associated Tcl procedure. By default, the new menu item gets added at the end of the parent menu. To place the new command at a specific location, specify the beforeMenu item option, which adds the new menu entry above the specificed entry.

On success of the command, the ID of the created action item is returned. If the command fails, the failure reason is returned as a string. Running the command on a previously associated menu item returns the ID of the associated action.

To enable a custom menu for a TOC page in a non-electric mode, execute the setActionEnabledInNonElectricMode command after running the addActionToMenu. To enable a custom menu for a read-only page, execute the setActionEnabledInReadOnlyState command after running addActionToMenu.

Return Type

INT

Syntax

```
addActionToMenu <parentMenu> <displayLabel> <command> <iconFile> <tooltip> <shortcut>
<context> ?beforeMenu?
```

Parameters

Parameter	Type	Description
`parentMenu`	`STRING`	Name of the menu in which this would be created as a sub-menu This parameter is required.
`displayLabel`	`STRING`	Label of the menu item This parameter is required.
`command`	`STRING`	Function to be called when the menu item is clicked This parameter is required.
`iconFile`	`STRING`	The path to the icon file. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32 and 48x48. This parameter is required.
`tooltip`	`STRING`	Tooltip text. This is currently unused This parameter is required.
`shortcut`	`STRING`	Shortcut sequence. This is currently unused This parameter is required.
`context`	`STRING`	The context in which this action is supported. For schematic canvas actions, this should be "sch" This parameter is required.
`beforeMenu`	`STRING`	This is a sub-menu in the parentMenuId list above which new menu will be added. If no value is passed or wrong value is passed the new menu will be added at the end of parentMenuId list. This parameter is optional. Default value is `None`.

Examples

```
#In the snippet below, the "Launch Text Editor" menu item is added to the "Tools" menu.
#The Tcl procedure launchCustomTextEditor is executed on clicking this menu item.
#The procedure determines the machine's platform and launches a text editor.
proc launchCustomTextEditor {} {
if { 1 != [string match -nocase "*win*" $::tcl_platform(platform)] } {
exec gvim &
} else {
exec notepad &
}
}
set icon [cps::getResourceFullPath {common/themes/light/24x24/Project.png}]
#This would return the ID of the newly created action. The ID can be used
#to get details about the action as follows:
set actionId [addActionToMenu Tools "Launch Text Editor" {launchCustomTextEditor} $icon
{} {} {sch}]
#This would return the action display label - "Launch Text Editor"
cps::getActionName sch $actionId


# To enable a custom menu for TOC page
addActionToMenu {Tools} { CustomMenuTOC } { launchCustomTextEditor } {} {} {} {sch}
set actionNonElectricMode [getActionId { sch } { CustomMenuTOC }]
setActionEnabledInNonElectricMode { sch } $actionNonElectricMode

# To enable a custom menu in the read-only mode
addActionToMenu {Tools} { CustomMenuReadOnly } { launchCustomTextEditor } {} {} {}
{sch}
set actionId4 [getActionId { sch } { CustomMenuReadOnly }]
setActionEnabledInReadOnlyState { sch } $actionId4

# custom menu will be added before Part Manager menu in Tools menu of menubar
addActionToMenu Tools "Launch Text Editor" {launchCustomTextEditor} $icon {} {} {sch}
"Part Manager"
```

Related Commands

addActionToMenuEx

addActionToMenuId

addActionToMenuIdEx

addSeparatorToMenu

getActionName

getResourceFullPath

createActionItem

getMenuId

setActionEnabledInNonElectricMode

setActionEnabledInReadOnlyState

addActionToMenuEx

Adds a menu entry in the parent menu. It is an extended version of the addActionToMenu command and allows specifying additional icons for "Selected" and "Hover" states of the menu item. If the "beforeMenu" item is specified, the new menu gets added above it, if no argument is passed or an incorrect argument is specified, the menu item gets added at the end of the parent menu. Clicking the menu entry executes the associated Tcl procedure. On success, the ID of the created action item is returned. If the command fails, the failure reason is returned as a string.

Return Type

INT

Syntax

```
addActionToMenuEx <parentMenu> <label> <command> <icon> <selectedIcon> <hoverIcon>
<tooltip> <shortcut> <context> ?beforeMenu?
```

Parameters

Parameter	Type	Description
`parentMenu`	`STRING`	Name of the menu within which this would be created as a sub-menu This parameter is required.
`label`	`STRING`	Label of the menu item This parameter is required.
`command`	`STRING`	Tcl procedure to be called when the menu item is clicked This parameter is required.
`icon`	`STRING`	The path to the icon file. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32, and 48x48. This parameter is required.

`selectedIcon`	`STRING`	The path to the icon to be displayed when the menu item is selected. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32, and 48x48. This parameter is required.
`hoverIcon`	`STRING`	The path to the icon to be displayed when the menu item is hovered. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32, and 48x48. This parameter is required.
`tooltip`	`STRING`	Tooltip text. This is currently unused and should be passed the value {} This parameter is required.
`shortcut`	`STRING`	Shortcut sequence. This is currently unused and should be passed the value {} This parameter is required.
`context`	`STRING`	The context in which this action is supported. For schematic canvas actions, this should be "sch" This parameter is required.
`beforeMenu`	`STRING`	This is a sub-menu in the parentMenuId list above which new menu will be added. If no value is passed or wrong value is passed the new menu will be added at the end of parentMenuId list. This parameter is optional. Default value is `None`.

Examples

```
#command to add the "Launch Text Editor" menu item in the "Tools" menu.
#the Tcl procedure launchCustomTextEditor is executed on clicking this menu item.
#the procedure determines the machine's platform and launches a text editor.
proc launchCustomTextEditor {} {
if { 1 != [string match -nocase "*win*" $::tcl_platform(platform)] } {
exec gvim &
} else {
exec notepad &
}
}

#command to load icons
set icon /home/user1/icons/icon1.png
set selectedIcon /home/user1/icons/icon2.png
set hoverIcon /home/user1/icons/icon3.png

#command to add the menu item to Tools
set actionId [addActionToMenuEx Tools "Launch Another Text Editor"
{launchCustomTextEditor} $icon $selectedIcon $hoverIcon {} {} {sch}]

#this would return the ID of the newly created action.
#the ID can be used to get details about the action as follows:
cps::getActionName sch $actionId
#this would return the action display label - "Launch Another Text Editor"

# custom menu will be added above Part Manager menu in Tools menu of menubar
addActionToMenu Tools "Launch Text Editor" {launchCustomTextEditor} $icon {} {} {sch}
"Part Manager"
```

Related Commands

addActionToMenu

addActionToMenuEx2

addActionToMenuId

addActionToMenuIdEx

addSeparatorToMenu

createActionItem

getMenuId

getActionName

getResourceFullPath

addActionToMenuEx2

Adds a menu entry in the parent menu. It is an extended version of the addActionToMenuEx command and allows specifying additional icons for "Selected" and "Hover" states of the menu item.

If "beforeMenu" item is specified, the new menu gets added above it ,else the menu item is added at the end of the parent menu. Clicking the menu entry would execute the associated Tcl procedure. On success, the ID of the created action item is returned. If the command fails, the failure reason is returned as a string.

Return Type

INT

Syntax

```
addActionToMenuEx2 <parentMenu> <actionName> <displayName> <command> <icon>
<selectedIcon> <hoverIcon> <tooltip> <shortcut> <context> ?beforeMenu?
```

Parameters

Parameter	Type	Description
`parentMenu`	`STRING`	Name of the menu within which this would be created as a sub-menu This parameter is required.
`actionName`	`STRING`	The name of the menu item to be added. This parameter is required.
`displayName`	`STRING`	The display name of the menu item to be added. This parameter is required.
`command`	`STRING`	Tcl procedure to be called when the menu item is clicked This parameter is required.

`icon`	`STRING`	The path to the icon file. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32, and 48x48. This parameter is required.
`selectedIcon`	`STRING`	The path to the icon to be displayed when the menu item is selected. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32, and 48x48. This parameter is required.
`hoverIcon`	`STRING`	The path to the icon to be displayed when the menu item is hovered. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32, and 48x48. This parameter is required.
`tooltip`	`STRING`	Tooltip text. This is currently unused and should be passed the value {} This parameter is required.
`shortcut`	`STRING`	Shortcut sequence. This is currently unused and should be passed the value {} This parameter is required.
`context`	`STRING`	The context in which this action is supported. For schematic canvas actions, this should be "sch" This parameter is required. Default value is `None`.
`beforeMenu`	`STRING`	This is a sub-menu in the parentMenuId list above which new menu will be added. If no value is passed or wrong value is passed the new menu will be added at the end of parentMenuId list. This parameter is optional. Default value is `None`.

Examples

```
#command to add the "Launch Text Editor" menu item in the "Tools" menu.
#the Tcl procedure launchCustomTextEditor is executed on clicking this menu item.
#the procedure determines the machine's platform and launches a text editor.
proc launchCustomTextEditor {} {
if { 1 != [string match -nocase "*win*" $::tcl_platform(platform)] } {
exec gvim &
} else {
exec notepad &
}
}

#command to load icons
set icon /home/user1/icons/icon1.png
set selectedIcon /home/user1/icons/icon2.png
set hoverIcon /home/user1/icons/icon3.png

#command to add the menu item to Tools just above Part Manager menu
set actionId [addActionToMenuEx2 Tools "Launch Another Text Editor"
{launchCustomTextEditor} $icon $selectedIcon $hoverIcon {} {} {sch} "Part Manager"]
#this would return the ID of the newly created action.
#the ID can be used to get details about the action as follows:
cps::getActionName sch $actionId
#this would return the action display label - "Launch Another Text Editor"
```

Related Commands

addActionToMenuEx

addActionToMenuId

addActionToMenuIdEx

addActionToMenuEx

addSeparatorToMenu

createActionItem

getMenuId

getActionName

getResourceFullPath

addActionToMenuId

It adds a menu entry in the parent menu specified by its ID. Clicking the menu entry executes the associated Tcl procedure. The menu item is added at the end. On success, the ID of the created action item is returned. On failure, the reason is returned as a string. Running this command on a previously associated menu item returns the ID of the associated action.

Return Type

INT

Syntax

```
addActionToMenuId <parentMenuId> <label> <command> ?<icon>? ?<tooltip>? ?<shortcut>? ?
<context>?
```

Parameters

Parameter	Type	Description
`parentMenuId`	`INT`	ID of the menu within which this would be created as a sub-menu This parameter is required.
`label`	`STRING`	Label of the menu item This parameter is required.
`command`	`STRING`	Tcl procedure to be called when the menu item is clicked This parameter is required.
`icon`	`STRING`	The path to the icon file. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32 and 48x48. This parameter is optional.
`tooltip`	`STRING`	Tooltip text. This is currently unused and should be passed the value {} This parameter is optional.
`shortcut`	`STRING`	Shortcut sequence. This is currently unused and should be passed the value {} This parameter is optional.
`context`	`STRING`	The context in which this action is supported. For schematic canvas actions, this should be "sch" This parameter is optional.

Examples

```
#in this example, an action is added to the "Help" menu that prints
#the Tcl version information in the Tcl command window.
set icon [cps::getResourceFullPath {common/themes/light/24x24/Project.png}]
addActionToMenuId [getMenuId "Help"] "Tcl Version" {puts "[info patchlevel]"} $icon {}
{} sch
```

Related Commands

getMenuId

getResourceFullPath

addActionToMenuEx

addActionToMenuId

addActionToMenuIdEx

getMenuId

createActionItem

addSeparatorToMenu

getActionName

addActionToMenuIdEx

It adds a menu entry in the parent menu denoted by the ID. This is an extended version of the addActionToMenuId command and allows specifying additional icons for "Selected" and "Hover" states of the menu item. The menu item is added at the end of the parent menu. Clicking the menu entry executes the associated Tcl procedure. On success, the ID of the created action item is returned. On failure, the reason for failure is returned as a string.

Return Type

INT

Syntax

```
addActionToMenuIdEx <parentMenuId> <label> <command> <icon> <selectedIcon> <hoverIcon>
<tooltip> <shortcut> <context>
```

Parameters

Parameter	Type	Description
`parentMenuId`	`INT`	ID of the menu within which this would be created as a sub-menu This parameter is required.
`label`	`STRING`	Label of the menu item This parameter is required.
`command`	`STRING`	Tcl procedure to be called when the menu item is clicked This parameter is required.
`icon`	`STRING`	The path to the icon file. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32 and 48x48. This parameter is required.

`selectedIcon`	`STRING`	The path to the icon to be displayed when the menu item is selected. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32 and 48x48. This parameter is required.
`hoverIcon`	`STRING`	The path to the icon to be displayed when the menu item is hovered. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32 and 48x48. This parameter is required.
`tooltip`	`STRING`	Tooltip text. This is currently unused and should be passed the value {} This parameter is required.
`shortcut`	`STRING`	Shortcut sequence. This is currently unused and should be passed the value {} This parameter is required.
`context`	`STRING`	The context in which this action is supported. For schematic canvas actions, this should be "sch" This parameter is required.

Examples

```
#in this example, an action is added to the "Help" menu that prints
#the Tcl version information in the Tcl command window.
#command to load icons
set icon /home/user1/icons/icon1.png
set selectedIcon /home/user1/icons/icon2.png
set hoverIcon /home/user1/icons/icon3.png
addActionToMenuIdEx [getMenuId "Help"] "Tcl Version" {puts "[info patchlevel]"} $icon
$selectedIcon $hoverIcon {} {} sch
```

Related Commands

addActionToMenuEx

addActionToMenuId

addSeparatorToMenu

createActionItem

getMenuId

getActionName

getResourceFullPath

addActionToMenuIdEx2

Adds a menu entry in the parent menu denoted by the parentMenuId. This is an extended version of the addActionToMenuId command and allows specifying additional icons for "Selected" and "Hover" states of the menu item. If "beforeMenu" is specified the new menu gets added above it, else the menu item is added at the end of the parent menu. Clicking the menu entry executes the associated Tcl procedure. On success, the ID of the created action item is returned. On failure, the reason for failure is returned as a string.

Return Type

INT

Syntax

```
addActionToMenuIdEx2 <parentMenuId> <label> <command> <icon> <selectedIcon> <hoverIcon>
<tooltip> <shortcut> <context> ?beforeMenu?
```

Parameters

Parameter	Type	Description
`parentMenuId`	`INT`	ID of the menu within which this would be created as a sub-menu This parameter is required.
`label`	`STRING`	Label of the menu item This parameter is required.
`command`	`STRING`	Tcl procedure to be called when the menu item is clicked This parameter is required.
`icon`	`STRING`	The path to the icon file. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32 and 48x48. This parameter is required.

`selectedIcon`	`STRING`	The path to the icon to be displayed when the menu item is selected. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32 and 48x48. This parameter is required.
`hoverIcon`	`STRING`	The path to the icon to be displayed when the menu item is hovered. Supported icon formats include JPEGs, PNGs, GIFs. Supported icon sizes in pixels are 12x12, 16x16, 24x24, 32x32 and 48x48. This parameter is required.
`tooltip`	`STRING`	Tooltip text. This is currently unused and should be passed the value {} This parameter is required.
`shortcut`	`STRING`	Shortcut sequence. This is currently unused and should be passed the value {} This parameter is required.
`context`	`STRING`	The context in which this action is supported. For schematic canvas actions, this should be "sch" This parameter is required.
`beforeMenu`	`STRING`	This is a sub-menu in the parentMenuId list above which new menu will be added. If no value is passed or wrong value is passed the new menu will be added at the end of parentMenuId list. This parameter is optional. Default value is `None`.

Examples

```
#in this example, an action is added to the "Help" menu that prints
#the Tcl version information in the Tcl command window.
#command to load icons
set icon /home/user1/icons/icon1.png
set selectedIcon /home/user1/icons/icon2.png
set hoverIcon /home/user1/icons/icon3.png
addActionToMenuIdEx2 [getMenuId "Help"] "Tcl Version" {puts "[info patchlevel]"} $icon
$selectedIcon $hoverIcon {} {} sch

# In this example, Custom Action named menu will be added in Help menu above its sub-
menu named Documentation.
addActionToMenuIdEx2 [getMenuId "Help"] "Custom Action" {puts "Custom Action"} $icon
$selectedIcon $hoverIcon {} {} "sch" "Documentation"
```

Related Commands

addActionToMenuEx

addActionToMenuId

addSeparatorToMenu

createActionItem

getMenuId

getActionName

getResourceFullPath

addActionToMenuIdEx

addActionToMenu

addMenuToContextMenuEx

Adds a pop-up menu within a context menu of the specified type and parent

Return Type

INT

Syntax

```
addMenuToContextMenuEx <contextMenuType> <subMenuName> <menuId>
```

Parameters

Parameter	Type	Description
`contextMenuType`	`INT`	The context menu type This parameter is required.
`subMenuName`	`STRING`	The sub-menu name of the context menu to which the menu should be added This parameter is required.
`menuId`	`INT`	The menu ID of the menu to be added to the context menu This parameter is required.

Examples

```
set wireContextMenuType [getContextMenuType "sch" {Wire} 0]

#Command to create menus
set contextMenu1Id [createMenuItem ContextMenu1 ContextMenu1 {} 1 1]
set contextMenu2Id [createMenuItem ContextMenu2 ContextMenu2 {} 1 1]

#Command to add menu to contextmenu. passing empty value in parent field adds the menu
to root menu item
addMenuToContextMenuEx $wireContextMenuType {} $contextMenu1Id

#Command to add Contextmenu2 to ContextMenu1
addMenuToContextMenuEx $wireContextMenuType ContextMenu1 $contextMenu2Id
```

Related Commands

deleteMenuFromContextMenu

deleteActionFromContextMenu

addSeparatorToContextMenu

deleteSeparatorFromContextMenu

deleteSeparatorFromToolBar

deleteMenuFromMenuName

deleteActionFromMenu

deleteMenuFromMenuBar

deleteToolItemFromToolBar

addActionToMenu

addActionToMenuId

addActionToMenuIdEx

addMenuToMenuId

addMenuToMenuName

addSeparatorToMenu

deleteSeparatorFromMenu

getMenuId

addMenuToMenuId

It adds a pop-up menu to an existing menu. It will fail if a menu or a pop-up menu already exists with the same name. It returns the ID of the pop-up menu on success and 0 on failure.

Return Type

INT

Syntax

```
addMenuToMenuId <parentMenuId> <menuName> <iconPath> <enabled>
```

Parameters

Parameter	Type	Description
`parentMenuId`	`INT`	Menu ID of the parent menu in which the sub-menu is added. Menu IDs of an existing menu can be obtained using the getMenuId command. This parameter is required.
`menuName`	`STRING`	The name of the pop-up menu to be added. This parameter is required.
`iconPath`	`STRING`	This parameter is reserved for future use and must be passed the empty list {}. This parameter is required.
`enabled`	`BOOLEAN`	Default state of the menu item. 1 enables the menu, and 0 disables the menu. This parameter is required.

Examples

```
#in this example, the pop-up menu "Launch Design Help" is created in the Help menu:
set parentMenuId [getMenuId "Help"]
set popUpMenuId 0
if { $parentMenuId != -1 } {
set popUpMenuId [addMenuToMenuId $parentMenuId "Launch Design Help" {} 1]
}
#trying to add a pop-up menu with the name of already existing pop-up menu fails
#and return a value of 0
addMenuToMenuId [getMenuId "View"] "Area Select Mode" {} 1
```

Related Commands

addActionToMenu

addActionToMenuId

addActionToMenuIdEx

addMenuToMenuId

addMenuToMenuName

addSeparatorToMenu

deleteSeparatorFromMenu

getMenuId

getResourceFullPath

addMenuToMenuName

It adds a pop-up menu to an existing menu. It will fail if a menu or pop-up menu already exists with the same name. It returns the ID of the pop-up menu on success and 0 on failure.

Return Type

INT

Syntax

```
addMenuToMenuName <parentMenuName> <menuName> <iconPath> <enabled>
```

Parameters

Parameter	Type	Description
`parentMenuName`	`STRING`	Name of the parent menu in which the sub-menu is added. Menu IDs of an existing menu can be obtained using the getMenuId command. This parameter is required.
`menuName`	`STRING`	The name of the pop-up menu to be added. This parameter is required.
`iconPath`	`STRING`	This parameter is reserved for future use and must be an empty list {}. This parameter is required.
`enabled`	`INT`	Default state of the menu item. 1 enables the menu, and 0 disables the menu. This parameter is required.

Examples

```
#in the following example, the pop-up menu "Launch Design Help" is created in
#the Help menu, and its ID is printed in the Tcl command window:
set popUpMenuId [addMenuToMenuName "Help" "Launch Design Help" {} 1]
puts "Menu id of the pop-up menu - $popUpMenuId"
#Trying to add a pop-up menu with the name of already existing
#pop-up menu fails and return a value of 0
addMenuToMenuName "View" "Area Select Mode" {} 1
```

Related Commands

addActionToMenu

addActionToMenuId

addActionToMenuIdEx

addMenuToMenuId

addMenuToMenuName

addSeparatorToMenu

deleteSeparatorFromMenu

getMenuId

getResourceFullPath

addSeparatorToContextMenu

It adds a separator between menu items within a context menu. The context menu is identified by the getContextMenuType command. The separator is added before the menu item specified in the beforeMenuItem parameter. It returns 1 on success and 0 on failure.

Return Type

INT

Syntax

```
addSeparatorToContextMenu <contextMenuType> <beforeMenuItem>
```

Parameters

Parameter	Type	Description
`contextMenuType`	`INT`	Type of the context menu returned by getContextMenuType function This parameter is required.
`beforeMenuItem`	`STRING`	Separator is added before the specified menu item This parameter is required.

Examples

```
#in the following example, an action is added to the context menu for
#single wire segments and a separator is added before it
set wireContextMenuType [getContextMenuType "sch" {Wire} 0]
set command {puts "Wire selected"}
set infoIcon {}
set menuLabel "Wire Sample"
#command to add the action to the context menu
addActionToContextMenu $wireContextMenuType $menuLabel $command {} {} {} sch
#command to add a separator before the newly added action
addSeparatorToContextMenu $wireContextMenuType $menuLabel
```

Related Commands

addActionToContextMenu

addActionToContextMenuEx

getContextMenuType

addSeparatorToMenu

It adds a separator before the specified menu item within the parent menu. It returns 1 on success and 0 on failure.

Return Type

INT

Syntax

```
addSeparatorToMenu <parentMenuName> <beforeMenuItem>
```

Parameters

Parameter	Type	Description
`parentMenuName`	`STRING`	Label of the parent menu This parameter is required.
`beforeMenuItem`	`STRING`	Label of the menu item before which the separator is added This parameter is required.

Examples

```
#this snippet adds a separator before the "Area Select Mode" menu
#item within the view menu.
addSeparatorToMenu {View} {Area Select Mode}
```

Related Commands

addActionToMenuEx

addActionToMenuId

addActionToMenuIdEx

getMenuId

createActionItem

getActionName

closeDialog

Closes the dialog whose handle has been provided. If a handle is not specified then the following error is returned:
"Wrong number of arguments :cps::closeDialog dialogHandler argument 1".
If the command is successful, it returns 0, otherwise, returns 1.

Return Type

INT

Syntax

```
cps::closeDialog <dialogHandler>
```

Parameters

Parameter	Type	Description
`dialogHandler`	`INT`	Handle of the dialog to be closed This parameter is required.

Examples

```
# This command returns the dialog handle of the mentioned dialog name.
set dh [sdaUI::openURLDialog CADENCE www.cadence.com testtab02 {} {0 0} {1000 400} 1 1]
puts $dh
# 82

# This command closes the dialog with the mentioned handle value
cps::closeDialog $dh
```

Related Commands

openURLDialog

createImportHierBlockDiffWidget

Return Type

INT

Syntax

```
sch::createImportHierBlockDiffWidget
```

Examples

```
sch::createImportHierBlockDiffWidget
#Output: 55
```

createMenuItem

Create a Menu Item

Return Type

INT

Syntax

```
cps::createMenuItem <menuName> <displayName> <iconPath> <isPopUp> <enabled>
```

Parameters

Parameter	Type	Description
`<menuName>`	`STRING`	Name of the menu This parameter is required.
`<displayName>`	`STRING`	Name of the menu item This parameter is required.
`<iconPath>`	`STRING`	Path for the icon This parameter is required.
`<isPopUp>`	`BOOL`	If the menu item is a popup or not This parameter is required.
`<enabled>`	`BOOL`	Enables status This parameter is required.

Examples

```
cps::createMenuItem {happy} {hap} {C:\Users\mraduls\Pictures\r.png} 1 1
#Output: 78010
```

createProject

Creates a new project based on an existing DE-HDL or OrCAD Capture project. This is how designs authored in other applications can be imported into System Capture. By default the schematic gets imported. For DE-HDL designs, the physical files can also be imported. When the command is successful, it returns 0 and opens the project automatically.

Return Type

BOOL

Syntax

```
createProject <project_path> -sourceProj <source_proj_path> -projName <project_name> -
srcBlock <srcBlockName> ?-designName <design_name>? ? -type<dehdl|capture>? ?-
layoutPath <source_layout_path>? ?-importPhysical <1|0>?
```

Parameters

Parameter	Type	Description
`project_path`	`STRING`	The absolute or relative path to the location where the project is to be created. This parameter is required.
`projName`	`STRING`	Name of the project to be created. The project CPM file gets created with this name. This parameter is required.
`sourceProj`	`STRING`	The path to the project file, DE-HDL or OrCAD Capture, from which the design has to be created This parameter is required.
`srcBlock`	`STRING`	The source top block name to be imported. If not specified, the value is fetched from the source design This parameter is optional.
`designName`	`STRING`	Name of the root design. Supported only for DE-HDL type projects This parameter is optional.
`type`	`STRING`	The source project type. Valid values are 'syscap' and 'dehdl'. This parameter is optional. Default value is `dehdl`.
`layoutPath`	`STRING`	The source design layout file path to be copied during design creation. Supported only for DE-HDL projects This parameter is optional.
`importPhysical`	`BOOL`	Include the the physical layout files associated with the source design. Supported only for DE-HDL projects This parameter is optional.

Examples

```
#command to create a new design from an existing DE-HDL project
#design with all arguments
createProject {d:/designs/designs_new/test_1} -sourceProj {D:/designs/dehdl/test.cpm} -
projName {test_1} -srcBlock {top} -type {dehdl}
#command to create a new design from an existing DE-HDL project
createProject {d:/designs/designs_new/test_1} -sourceProj {D:/designs/dehdl/test.cpm} -
projName {test_1}
#command to create a new design from an existing DE-HDL with changed design name
createProject {./syscap_design} -sourceProj {./src_design/src_design.cpm} -projName
{syscap_design} -srcBlock {top} -designName {test-top} -type {dehdl}
#command to create a new design from an existing DE-HDL with a changed design name and
also importing a layout file
createProject {./test1} -sourceProj {./test/test.cpm} -projName {top} -srcBlock {} -
designName {renamed-block} -type {dehdl} -layoutPath
{../../test/worklib/top/physical/920-10553.brd}
#command to create a new design from an existing DE-HDL with a changed design name and
importing the physical data associated with the design block
createProject {./test1} -sourceProj {./test/test.cpm} -projName {top} -srcBlock {} -
designName {renamed-block} -type {dehdl} -importPhysical 1
#command to create a new design from an existing OrCAD Capture design
createProject {d:/designs/designs_new/test_1} -sourceProj {D:/designs/capture/test.dsn}
-projName {test_1}
```

Related Commands

newProject

createTab

This command creates a new tab widget in the specified workspace and returns the widget handle.

Return Type

INT

Syntax

```
cps::createTab <title> <id> <workspaceName> <shouldShowTabMenu>
```

Parameters

Parameter	Type	Description
`title`	`STRING`	Specify title of the tab This parameter is required.
`id`	`STRING`	Specify id of the tab This parameter is required.
`workspaceName`	`STRING`	Specify name of the workspace This parameter is required.
`shouldShowTabMenu`	`BOOL`	Specify value to show(true/false) tab menu This parameter is required.

Examples

```
#Command to create a tab widget in workspace 1
cps::createTab TAB_A 1 1 true

#If any argument is missing then it gives an error as "Wrong number of
#arguments :cps::createTab title id workspaceName shouldShowTabMenu argument 1"
cps::createTab
```

deleteActionFromContextMenu

Deletes an action from context menu of a given type and parent

Return Type

INT

Syntax

```
deleteActionFromContextMenu <contextMenuType> <actionName> <parentMenuName>
```

Parameters

Parameter	Type	Description
`contextMenuType`	`INT`	context menu type This parameter is required.
`actionName`	`STRING`	name of the action to be deleted This parameter is required.
`parentMenuName`	`STRING`	parent menu name from which the action should be deleted This parameter is optional.

Examples

```
set wireContextMenuType [getContextMenuType "sch" {Wire} 0]

#Command to create menus
set contextMenu1Id [createMenuItem ContextMenu1 ContextMenu1 {} 1 1]

#Command to add menu to contextmenu. passing empty value in parent field adds the menu
to root menu item
addMenuToContextMenuEx $wireContextMenuType {} $contextMenu1Id
addActionToContextMenuEx $wireContextMenuType TestAct sampleHandler1 {} {} {} TestAct
{} sch ContextMenu1

deleteActionFromContextMenu $wireContextMenuType TestAct ContextMenu1
#the above command deletes TestAct from ContextMenu
```

Related Commands

addMenuToContextMenuEx

deleteMenuFromContextMenu

addSeparatorToContextMenu

deleteSeparatorFromContextMenu

deleteSeparatorFromToolBar

deleteMenuFromMenuName

deleteActionFromMenu

deleteMenuFromMenuBar

deleteToolItemFromToolBar

addActionToMenu

addActionToMenuId

addActionToMenuIdEx

addMenuToMenuId

addMenuToMenuName

addSeparatorToMenu

deleteSeparatorFromMenu

getMenuId

deleteActionFromMenu

Deletes an action name from the given parent menu and context.

Return Type

INT

Syntax

```
deleteActionFromMenu <parentMenuName> <actionName> <context>
```

Parameters

Parameter	Type	Description
`parentMenuName`	`STRING`	Name of the parent menu from which the action should be deleted This parameter is required.
`actionName`	`STRING`	Name of the action to be deleted This parameter is required.
`context`	`STRING`	The context to which the action belongs. if no context is given default context 'sch' is considered This parameter is optional.

Examples

```
#Command to delete the action 'Part Developer' from menu Tools which belongs to context
sch
deleteActionFromMenu {Tools} {Part Developer} {sch}
```

Related Commands

deleteMenuFromContextMenu

deleteActionFromContextMenu

addSeparatorToContextMenu

deleteSeparatorFromContextMenu

deleteSeparatorFromToolBar

deleteMenuFromMenuName

deleteMenuFromMenuBar

deleteToolItemFromToolBar

addActionToMenu

addActionToMenuId

addActionToMenuIdEx

addMenuToMenuId

addMenuToMenuName

addSeparatorToMenu

deleteSeparatorFromMenu

getMenuId

deleteMenuFromContextMenu

Deletes a menu from the context menu of a given type and parent. If the parentMenuName is not provided, it deletes the menu item from the top level context menu identified by the contextMenuType parameter.

Return Type

INT

Syntax

```
deleteMenuFromContextMenu <contextMenuType> <menuName> ?<parentMenuName>?
```

Parameters

Parameter	Type	Description
`contextMenuType`	`INT`	The context menu type returned by the getContextMenuType command This parameter is required.
`menuName`	`STRING`	Name of the menu to be deleted This parameter is required.
`parentMenuName`	`STRING`	Parent menu name from which the menu should be deleted This parameter is optional. Default value is `{}`.

Examples

```
set wireContextMenuType [getContextMenuType "sch" {Wire} 0]

#Command to create menus
set contextMenu1Id [createMenuItem ContextMenu1 ContextMenu1 {} 1 1]
set contextMenu2Id [createMenuItem ContextMenu2 ContextMenu2 {} 1 1]
addMenuToContextMenuEx $wireContextMenuType {} $contextMenu1Id
addMenuToContextMenuEx $wireContextMenuType ContextMenu1 $contextMenu2Id

deleteMenuFromContextMenu $wireContextMenuType ContextMenu2 ContextMenu1
#This deletes ContextMenu2 from ContextMenu1

deleteMenuFromContextMenu $wireContextMenuType ContextMenu1 {}
#This deletes ContextMenu1 from the root context menu
```

Related Commands

addMenuToContextMenuEx

addActionToMenu

addActionToMenuId

addActionToMenuIdEx

addMenuToMenuId

addMenuToMenuName

addSeparatorToMenu

deleteSeparatorFromMenu

getMenuId

getResourceFullPath

deleteMenuFromMenuName

Deletes a menu option from a menu

Return Type

INT

Syntax

```
deleteMenuFromMenuName <parentMenuName> <childMenuName>
```

Parameters

Parameter	Type	Description
`parentMenuName`	`STRING`	Name of the menu from which an option should be deleted This parameter is required.
`childMenuName`	`STRING`	Name of the menu option to be deleted This parameter is required.

Examples

```
#Command to delete the menu option "Draw Wire" from 'Place' menu
deleteMenuFromMenuName {Place} {Draw Wire}
```

Related Commands

addMenuToContextMenuEx

deleteMenuFromContextMenu

deleteActionFromContextMenu

addSeparatorToContextMenu

deleteSeparatorFromContextMenu

deleteSeparatorFromToolBar

deleteActionFromMenu

deleteMenuFromMenuBar

deleteToolItemFromToolBar

addActionToMenu

addActionToMenuId

addActionToMenuIdEx

addMenuToMenuId

addMenuToMenuName

addSeparatorToMenu

deleteSeparatorFromMenu

getMenuId

deleteSeparatorFromContextMenu

Deletes the separator between menu items within a context menu. The context menu is identified by the 'getContextMenuType' command. The separator is deleted before the menu item specified in the 'beforeMenuItem' parameter and the parent is specified by 'parentMenu' paramenter. Returns 1 on success and 0 on failure.

Return Type

INT

Syntax

```
deleteSeparatorFromContextMenu <contextMenuType> <beforeMenuItem> ?<parentMenu>?
```

Parameters

Parameter	Type	Description
`contextMenuType`	`INT`	Context menu type This parameter is required.
`beforeMenuItem`	`STRING`	Name of the menu option before which the separator should be added This parameter is required.
`parentMenu`	`STRING`	Name of menu to which the beforeMenuItem belongs This parameter is optional.

Examples

```
set wireContextMenuType [getContextMenuType "sch" {Wire} 0]
set contextMenu1Id [createMenuItem ContextMenu1 ContextMenu1 {} 1 1]

#passing empty value in parent field adds the menu to root menu item
addMenuToContextMenuEx $wireContextMenuType {} $contextMenu1Id
addActionToContextMenuEx $wireContextMenuType TestAct sampleHandler1 {} {} {} TestAct
{} sch ContextMenu1
addSeparatorToContextMenu $wireContextMenuType TestAct ContextMenu1

#Command to delete the separator before action TestAct, which belongs to ContextMenu1
deleteSeparatorFromContextMenu $wireContextMenuType TestAct ContextMenu1

#Command to delete the separator before action 'Add Special', which belongs to
#root context menu of the 'Wire' menu type
deleteSeparatorFromContextMenu $wireContextMenuType {Add Special} {}
```

Related Commands

addMenuToContextMenuEx

deleteMenuFromContextMenu

deleteActionFromContextMenu

addSeparatorToContextMenu

deleteSeparatorFromToolBar

deleteMenuFromMenuName

deleteActionFromMenu

deleteMenuFromMenuBar

deleteToolItemFromToolBar

addActionToMenu

addActionToMenuId

addActionToMenuIdEx

addMenuToMenuId

addMenuToMenuName

addSeparatorToMenu

deleteSeparatorFromMenu

getMenuId

deleteSeparatorFromMenu

It deletes a separator from a menu.

Return Type

INT

Syntax

```
deleteSeparatorFromMenu <menuId> <menuName> <beforeAction>
```

Parameters

Parameter	Type	Description
`<menuId>`	`INT`	ID of the menu from where separator needs to be deleted This parameter is required.
`<menuName>`	`STRING`	Name of the menu This parameter is required.
`<beforeAction>`	`STRING`	It signifies before which menu separator needs to be deleted This parameter is required.

Examples

```
deleteSeparatorFromMenu 78004 {Reports} {Component-Net Connections}
```

deleteSeparatorFromToolBar

Deletes the separator after a toolbar item and section from a toolbar, which belongs to a given context.

Return Type

INT

Syntax

```
deleteSeparatorFromToolBar <context> <section> <afterMenuItem>
```

Parameters

Parameter	Type	Description
`context`	`STRING`	Context to which the toolbar belongs This parameter is required.
`section`	`INT`	The toolbar section from which the separator should be removed This parameter is required.
`afterMenuItem`	`STRING`	The toolbar item name This parameter is required.

Examples

```
#Command to delete the separator from the toolbar after the toolbar item 'Cut'
deleteSeparatorFromToolBar {sch} 1 {Cut}
```

Related Commands

addMenuToContextMenuEx

deleteMenuFromContextMenu

deleteActionFromContextMenu

addSeparatorToContextMenu

deleteSeparatorFromContextMenu

deleteMenuFromMenuName

deleteActionFromMenu

deleteMenuFromMenuBar

deleteToolItemFromToolBar

addActionToMenu

addActionToMenuId

addActionToMenuIdEx

addMenuToMenuId

addMenuToMenuName

addSeparatorToMenu

deleteSeparatorFromMenu

getMenuId

deleteToolItemFromToolbar

Deletes a toolbar item from the given section and context of a toolbar

Return Type

INT

Syntax

```
deleteToolItemFromToolBar <toolItemName> <section> ?<context>?
```

Parameters

Parameter	Type	Description
`toolItemName`	`STRING`	The name of the toolbar item to be deleted This parameter is required.
`section`	`INT`	The section to which the toolbar item belongs This parameter is required.
`context`	`STRING`	The context to which the toolbar belongs. If no context is specified, the default context 'sch' is used. This parameter is optional.

Examples

```
#Command to delete the toolbar item 'Draw Wire' from section one
deleteToolItemFromToolBar {Draw Wire} 1 {sch}
```

Related Commands

addMenuToContextMenuEx

deleteSeparatorFromContextMenu

deleteSeparatorFromToolBar

deleteMenuFromMenuName

deleteMenuFromMenuBar

addActionToMenu

addActionToMenuId

addActionToMenuIdEx

addMenuToMenuId

addMenuToMenuName

addSeparatorToMenu

deleteSeparatorFromMenu

getMenuId

dockedHybridInit

This command returns the hybrid widget handle of the widget hosting the mentioned URL.

Return Type

INT

Syntax

```
cpCommon::dockedHybridInit <title> <widgetName> <URL>
```

Parameters

Parameter	Type	Description
`title`	`STRING`	Title of the widget This parameter is required.
`widgetName`	`STRING`	Name of the widget. This name can be used for looking up the component handle using cps::findComponentByName This parameter is required.
`URL`	`STRING`	URL of the content to host This parameter is required.

Examples

```
# This creates a hybrid widget handle for the URL hosting widget.
set hybridHandle [cpCommon::dockedHybridInit {CADENCE} {testtab02} {www.cadence.com}]
puts $hybridHandle
# This command returns an integer value for example:
# 92
```

Related Commands

addDock

setDockedWidgetVisibility

setDockedWidgetVisibilityOff

drawWire

Draws a wire, a bus or a netgroup on the schematic page.

Return Type

NONE

Syntax

```
drawWire ?-netgroup? {<signalName>} [list <ptX1>,<ptY1> <ptX2>,<ptY2> ...] ?<lsb>
<msb>?
```

Parameters

Parameter	Type	Description
`-netgroup`	`STRING`	Type of wire is NetGroup This parameter is optional.
`{<signalName>}`	`STRING`	Name of the signal. The signalName is recommended to be specified inside { } (curly brackets) to take care of the whitespace or any other special character in signal name. This parameter is required.
`[list <ptX>, <ptY> ...]`	`INTEGER`	The list of XY point pairs of the wire points. A minimum of two point pairs must be specified. This parameter is required.
`<lsb> <msb>`	`INTEGER`	The Least Significant Bit (LSB) and the Most Significant Bit (MSB) for bus wire. For scalar or netgroup wire, omit these parameters or specify -1 This parameter is optional.

Examples

```
drawWire {_N14} [list 5100,8500 12600,8500] -1 -1
```

```
drawWire {CLK} [list 5100,8600 12600,8600]
```

```
drawWire {ADDR&lt;15..0&gt;} [list 5100,8700 12600,8700] 0 15
```

```
drawWire -netgroup {NG1} [list 5100,2600 14000,2600 14000,2600] -1 -1
```

getAutomationMode

Returns the current state of automation. Return value should be compared against the following Tcl variables:

cps::eCPS_AUTOMATION_ENABLED
cps::eCPS_AUTOMATION_DISABLED
cps::eCPS_AUTOMATION_NOTSET

Enabling automation suppresses message boxes. For more information on this command refer cps::setAutomationMode

Return Type

INT

Syntax

```
cps::getAutomationMode
```

Examples

```
set mode [cps::getAutomationMode]
if { $::cps::eCPS_AUTOMATION_NOTSET == $mode } {
puts "Automation mode is not set"
} elseif { $::cps::eCPS_AUTOMATION_ENABLED == $mode } {
puts "Automation mode is enabled. Message boxes and dialogs are being suppressed"
} elseif { $::cps::eCPS_AUTOMATION_DISABLED == $mode } {
puts "Automation mode is disabled. Message boxes and dialogs are being shown"
}

puts $::cps::eCPS_AUTOMATION_ENABLED
# prints 1

puts $::cps::eCPS_AUTOMATION_DISABLED
# prints 2

puts $::cps::eCPS_AUTOMATION_NOTSET
# prints 0
```

Related Commands

setAutomationMode

getCheckedOutLicense

Returns the license string checked out by the user

Return Type

NONE

Syntax

```
cps::getCheckedOutLicense
```

Examples

```
set lic [cps::getCheckedOutLicense]
puts $lic
# prints Designer_System_Capture
```

getContextMenuType

Use getContextMenuType to get the ID of the context menu that is displayed when objects of the types defined by the objectTypes list are selected in the editor. Context menus are shown on right mouse button (RMB) press. This ID can be used to modify the context menu by adding new actions or pop-up menus to it or by removing items from it.

Return Type

INT

Syntax

```
getContextMenuType <context> <objectTypes> ?isMulti?
```

Parameters

Parameter	Type	Description
`context`	`STRING`	Editor context in which the context menu is applicable, example "sch" for Schematic Editor This parameter is required.
`objectTypes`	`LIST`	List of strings denoting the object types from which this context menu is displayed. This parameter is required.
`isMulti`	`BOOLEAN`	Flag indicating whether the context menu type being retrieved applies to multiple objects of the same type. By default this is false. This parameter is optional.

Examples

```
set menuType [getContextMenuType sch {Route}]
#Returns the ID of the context menu that is displayed on RMB,
#when a single route object is selected on the schematic canvas. “Routes”
#denote wires, buses, and connect lines.
set menuType [getContextMenuType sch {Route} 1]
#Returns the ID of the context menu that is displayed on RMB,
#when more than one route objects are selected on the schematic canvas.
set menuType [getContextMenuType sch {Route InstTerm}]
#Returns the ID of the context menu that is displayed on RMB,
#when both route and instterms are select on the canvas.
```

Related Commands

addActionToContextMenu

addActionToContextMenuEx

addSeparatorToContextMenu

getEditorAttributes

Returns the current theme and color codes for schematic color, canvas background color, and the grid lines color in JSON format. Also returns the color code for black, dark, light, mid background color code in both the themes in the following format:

```
{
"Application Dark Theme Color Code": "#222830",
"Application Light Theme Color Code": "white",
"Canvas Background Black Color Code": "#000000",
"Canvas Background Dark Color Code": "#808080",
"Canvas Background Light Color Code": "#b3b3b3",
"Canvas Background Mid Color Code": "#999999",
"Canvas Background Theme": "Light",
"Grid Lines Color": "#383838",
"Schematic Black theme Color Code": "#000000",
"Schematic Theme": "light"
}
```

Return Type

STRING

Syntax

```
sch::getEditorAttributes
```

Examples

```
sch::getEditorAttributes
```

getItemTypeByViewId

Returns the item type of the tab for the specified view ID

Return Type

STRING

Syntax

```
cps::getItemTypeByViewId <viewid>
```

Parameters

Parameter	Type	Description
`viewid`	`STRING`	Tab view ID for which item type is required This parameter is required.

Examples

```
#To print the item type of the tab opened at the second index in the main window.
set tabCount [cps::getTabViewCount]
if { $tabCount &gt; 2 } {
set viewId [cps::getTabViewIdByIndex 1]
set itemType [cps::getItemTypeByViewId $viewId]
puts "Item type of tab is $itemType"
}
```

Related Commands

getTabViewCount

getTabViewIdByIndex

getViewTypeByViewId

getTabViewIds

getLanguageInfo

Returns the following information of the system:
1. Language
2. Country name
3. Language code
4. Country code
5. Native language name
6. Native country name

Return Type

STRING

Syntax

```
getLanguageInfo
```

Examples

```
#For a system where the locale is set to English(India)

cps::getLanguageInfo
{country India} {language English} {languageCode en} {languagePrefix en_IN}
{nativeCountry India} {nativeLanguage English}
```

getMenuId

It returns the ID of the menu given its label. The ID of the menu uniquely identifies the menu object and can be used in other commands. Refer to the examples for sample usage.

Return Type

INT

Syntax

```
getMenuId <menuName>
```

Parameters

Parameter	Type	Description
`menuName`	`STRING`	Menu label This parameter is required.

Examples

```
#command to get the menu ID of the "Help" menu and add a menu item in it.
set helpMenuId [getMenuId "Help"]
set icon [cps::getResourceFullPath {common/themes/light/24x24/Project.png}]
addActionToMenuId $helpMenuId "Tcl Version" {puts "[info patchlevel]"} $icon {} {} sch
```

Related Commands

addActionToMenuId

addActionToMenuIdEx

addActionToMenu

getMenuName

getNewProjectPath

Returns the default path where new projects would be created. The path is determined based on the following rules, listed in order of priority:

1. Value of the SYSTEM_CAPTURE_NEW_PROJECT_PATH environment variable, if defined
2. Path to the application's current working directory, if write permissions are available
3. Path where the last project was created
4. Path to the user's home directory

Return Type

STRING

Syntax

```
cps::getNewProjectPath
```

Examples

```
cps::getNewProjectPath
# returns the project directory path after computing the locations mentioned in the
description
```

getProject

Returns the absolute path of the currently open project.

Return Type

STRING

Syntax

```
cps::getProject
```

Examples

```
cps::getProject
# return the project file such as D:/designs/test/logic/test.sdax/test.cpm
```

Related Commands

getProjectDir

getProjectCPM

It returns the cpm filename of the opened project.

Return Type

STRING

Syntax

```
cps::getProjectCPM
```

Examples

```
cps::getProjectCPM
#output: inverter.cpm
```

getProjectDir

It returns the absolute path of the folder of the project opened.

Return Type

STRING

Syntax

```
cps::getProjectDir
```

Examples

```
cps::getProjectDir
```

getProjectMainCPM

It returns the absolute path of the opened project file.

Return Type

STRING

Syntax

```
cps::getProjectMainCPM
```

Examples

```
cps::getProjectMainCPM
#output: C:/Users/sathiyan/Project/radio/AFAmplifier/afamplifier.cpm
```

getProjectMainDir

It returns the folder path of the opened project.

Return Type

STRING

Syntax

```
cps::getProjectMainDir
```

Examples

```
cps::getProjectMainDir
#output: C:/Users/sathiyan/Project/radio/AFAmplifier
```

getProperty

This command is used to query the properties of hybrid dialog.

Return Type

STRING

Syntax

```
cps::getProperty <componentHandle> <propertyName>
```

Parameters

Parameter	Type	Description
`componentHandle`	`INT`	Component handle This parameter is required.
`propertyName`	`STRING`	Dialog property name This parameter is required.

Examples

```
set ret [sdaUI::openURLTab "cadence home page" "www.cadence.com" 1 ""]
#the above command stores the handle of the component in ret
cps::getProperty $ret visible
#the above command queries the visible property of the component
#returns true if the component is visible otherwise false
```

Related Commands

setProperty

openURLTab

openURLDialog

hybridDialog

getPulseDirectiveValue

Get the Pulse Directive Value for design Attributes of currently opened design in System Capture or Desktop Workflow

Return Type

Value

Syntax

```
cps::getPulseDirectiveValue CUSTOMVAR <name>
```

Parameters

Parameter	Type	Description
`name`	`String`	Name of design attribute This parameter is required.

Examples

```
cps::getPulseDirectiveValue CUSTOMVAR ProgramName

#Output is new
```

getTabViewCount

Returns the number of tabs opened in the primary workspace.

Return Type

INT

Syntax

```
cps::getTabViewCount
```

Examples

```
#Three tabs are currently opened in the primary workspace.
cps::getTabViewCount
# returns 3
```

Related Commands

getTabViewCountEx

getTabViewIdByIndex

getTabViewIdByIndexEx

getTabViewCountEx

Returns the number of tabs opened in the specified workspace.

Return Type

INT

Syntax

```
cps::getTabViewCountEx <workspace>
```

Parameters

Parameter	Type	Description
`workspace`	`STRING`	Name of workspace for which count of opened tabs is required This parameter is required.

Examples

```
#To print the count of tabs for each workspace that has design pages opened.
set pageListDict [ schPageUtils::getDesignPagesAllInfo ]
set size [schPageUtils::getPagesInfoSize $pageListDict ]
for {set i 1} {$i &lt;= $size} {incr i} {
set viewId [schPageUtils::getPageId $pageListDict $i]
set ws [cps::getWorkspaceHostingTabWidget $viewId ]
set wsCount [cps::getTabViewCountEx $ws]
}
```

Related Commands

getTabViewCount

getWorkspaceHostingTabWidget

getTabViewIdByIndex

getTabViewIdByIndexEx

getTabViewIdByIndex

Returns the tab view ID for the given tab index in the primary workspace. An empty string is returned if provided tab index does not exist.

Return Type

STRING

Syntax

```
cps::getTabViewIdByIndex <index>
```

Parameters

Parameter	Type	Description
`index`	`INT`	Tab index for which view ID is required This parameter is required.

Examples

```
#To print the view ID of the tab opened at the second index in the main window.
set tabCount [cps::getTabViewCount]
if { $tabCount &gt; 2 } {
set viewId [cps::getTabViewIdByIndex 1]
puts "View Id for second tab is $viewId"
}
```

Related Commands

getTabViewCount

getTabViewIdByIndexEx

Returns the tab view ID for the given tab index in the specified workspace name. An empty string is returned if the specified tab index does not exist in the workspace.

Return Type

STRING

Syntax

```
cps::getTabViewIdByIndexEx <index> <workspace>
```

Parameters

Parameter	Type	Description
`index`	`INT`	Tab index for which the view ID is required This parameter is required.
`workspace`	`STRING`	Workspace that has the tab opened for which view ID is required This parameter is required.

Examples

```
#To print the view ID of the tab opened at the first index in WorkSpace1
set tabCount [cps::getTabViewCountEx "WorkSpace1"]
if { $tabCount &gt; 1 } {
set viewId [cps::getTabViewIdByIndex 0]
puts "In WorkSpace1 view Id for first tab is $viewId "
}
```

getTabViewIds

Returns a list of all the opened tab view IDs for a given view type and item type. All opened tabs would be returned if the view type and item type provided are empty strings.

Return Type

LIST

Syntax

```
cps::getTabViewIds <viewtype> <itemtype>
```

Parameters

Parameter	Type	Description
`viewtype`	`STRING`	View type for which all open tab view IDs are required This parameter is required.
`itemtype`	`STRING`	Item type of the tab for which the tab view IDs are required This parameter is required.

Examples

```
#To print the view IDs of all the tabs with view and item type same as the tab opened
at the second index in main window.
set tabCount [cps::getTabViewCount]
if { $tabCount &gt; 2 } {
set viewId [cps::getTabViewIdByIndex 1]
set viewType [cps::getViewTypeByViewId $viewId]
set itemType [cps::getItemTypeByViewId $viewId]
set tabs [cps::getTabViewIds $viewType $itemType]
puts "Opened tabs for View type $viewType and Item type $itemType are $tabs"
}

# List all the opened schematic tabs:
set tabs [ cps::getTabViewIds SCH PAGE ]
puts "Opened schematic tabs are $tabs"
```

Related Commands

getTabViewCount

getTabViewIdByIndex

getItemTypeByViewId

getItemTypeByViewId

getTheme

Returns the name of the current application theme.

Return Type

STRING

Syntax

```
cps::getTheme
```

Examples

```
puts [cps::getTheme]
#prints "dark" or "light"
```

getViewTypeByViewId

Returns the view type for the tab whose view ID is specified.

Return Type

STRING

Syntax

```
cps::getViewTypeByViewId <viewId>
```

Parameters

Parameter	Type	Description
`viewId`	`STRING`	View ID of tab for which view type is required This parameter is required.

Examples

```
#To print the view type of the tab opened at the second index in the main window.
set tabCount [cps::getTabViewCount]
if { $tabCount &gt; 2 } {
set viewId [cps::getTabViewIdByIndex 1]
set viewType [cps::getViewTypeByViewId $viewId]
puts "View type of tab is $viewType"
}
```

Related Commands

getTabViewCount

getTabViewIdByIndex

getItemTypeByViewId

getTabViewIds

getWorkspaceHostingTabWidget

Returns the workspace name where the tab with the specified view ID is opened. Returns an empty string if the tab is not open.

Return Type

STRING

Syntax

```
cps::getWorkspaceHostingTabWidget <viewId>
```

Parameters

Parameter	Type	Description
`viewId`	`STRING`	View ID of the tab for which hosting workspace name is required This parameter is required.

Examples

```
#To print the workspace name for all the pages in a design.
set pageListDict [ schPageUtils::getDesignPagesAllInfo ]
set size [schPageUtils::getPagesInfoSize $pageListDict ]
for {set i 1} {$i &lt;= $size} {incr i} {
set viewId [schPageUtils::getPageId $pageListDict $i]

set ws [cps::getWorkspaceHostingTabWidget $viewId ]
if { $ws != {} } {
puts "Workspace $ws has tab $viewId"
}
}
```

Related Commands

getDesignPagesAllInfo

getPageId

hasTabWidget

getTabViewCountEx

hasTabWidget

Checks if a tab with the given ID is open.

Return Type

BOOL

Syntax

```
cps::hasTabWidget <viewId>
```

Parameters

Parameter	Type	Description
`viewId`	`STRING`	View ID for the tab widget This parameter is required.

Examples

```
#To print the open tab status for all pages in design.
set pageListDict [ schPageUtils::getDesignPagesAllInfo ]
set size [schPageUtils::getPagesInfoSize $pageListDict ]
for {set i 1} {$i &lt;= $size} {incr i} {
set viewId [schPageUtils::getPageId $pageListDict $i]
set openStatus [cps::hasTabWidget $viewId]
puts "For View $viewId - Tab Open Status: $openStatus"
}
```

Related Commands

getDesignPagesAllInfo

getPageId

help

Launches Cadence Help Documentation Viewer with details about the Tcl command being searched for.

Return Type

NONE

Syntax

```
help <command>
```

Examples

```
help addMenuToMenu
```

hideitem

Hides the specified page. If the page is already hidden, it gets displayed. You can use the sch::dbGetActivePageSpath command to get the Page path. If an incorrect page path is specified, failure is reported with a return value of 1. A return value of 0 denotes success.

Return Type

ret

Syntax

```
hideItem <item_id> <item_type> <view_type> <hide_option>
```

Parameters

Parameter	Type	Description
`item_id`	`STR`	Path of the page to be hidden or displayed This parameter is optional. Default value is `active page path`.
`item_type`	`STR`	item type This parameter is optional. Default value is `active item type`.
`view_type`	`STR`	view type This parameter is optional. Default value is `active view type`.
`hide_option`	`INT`	hide option This parameter is optional. Default value is `1`.

Examples

```
hideItem {@worklib.aligntest(tbl_1):Page(2)} {SCH} {PAGE} {1} // hide page , nothing
happens if it is already hidden
hideItem {@worklib.aligntest(tbl_1):Page(2)} {SCH} {PAGE} {0} // unhide page , nothing
happens if it is already visible
```

Related Commands

dbGetActivePageSpath

hideitem

isCaptureProject

Returns 1 or 0 based upon the condition whether the currently active project is a Capture library tagged project or not

Return Type

BOOLEAN

Syntax

```
cps::isCaptureProject
```

Examples

```
For a Capture library tagged project-
cps::isCaptureProject
returns 1

For a DE-HDL library tagged project-
cps::isCaptureProject
returns 0
```

launchAllegro

Launches Allegro PCB Layout editor

Return Type

INT

Syntax

```
launchAllegro
```

launchFileBrowser

This command opens a dialog box with a title. File type filters, labels, and extension wildcards can be specified as shown in the examples section. The full path of the selected file is returned.

Return Type

STRING

Syntax

```
cps::launchFileBrowser <title> <initial directory> <filter label> <wildcard>
```

Parameters

Parameter	Type	Description
`title`	`STRING`	Name shown on the file chooser title bar This parameter is required.
`initial directory`	`STRING`	Directory to open on launching file chooser dialog This parameter is required.
`filter label`	`STRING`	Label to be shown in the file type field of a platform file dialog This parameter is required.
`wildcard`	`STRING`	Wildcard regular expression to filter files This parameter is required.

Examples

```
cps::launchFileBrowser "Select Voltage Rules File" /home/jdoe {Voltage Rules} {*.vrl}
```

logToSessionWindow

It displays the given message in the session window. The message will be displayed in the session log window after processing. Severity levels 1-10 can be used for different message priorities, although 2=Warning, 3= Error, 4=Fatal error are the only possible values now. Rest will be supported in future release.

Return Type

NONE

Syntax

```
cps::logToSessionWindow <logMsg> <src> <type>
```

Parameters

Parameter	Type	Description
`logMsg`	`STRING`	Message to be displayed in session window This parameter is required.
`src`	`STRING`	Initiator of the message, its value could be "". This field gives information about the tool which initiated the message. This will be supported in future release. This parameter is required.
`type`	`INT`	Severity of the message. Its possible values are 2 = Warning, 3= Error, 4 = Fatal error This parameter is required.

Examples

```
#command to display the appversion in session window
cps::logToSessionWindow "appversion=[cps::getAppVersion]" "" 2
#output: Returns none
```

mapName

This function performs namespace mapping from an input domain to an output domain. The function of this Tcl command is equivalent to the standalone executable nmp. The allowed namespaces are: Ascii, CDBA, Concept, Def, Gcf, Genesis, Lef, Library, Print, Sdf, Spf, Spectre, SpectreHDL, Spef, Spice, SysVerilog, VHDL, VHDL87, Verilog, VerilogA, VerilogAMS, VHDLAMS.

Return Type

STRING

Syntax

```
cps:mapName <inputName> <fromNamespace> <toNamespace>
```

Parameters

Parameter	Type	Description
`inputName`	`STRING`	Name to be converted This parameter is required.
`fromNamespace`	`STRING`	From namespace in nmp This parameter is required.
`toNamespace`	`STRING`	To namespace in nmp This parameter is required.

Examples

```
#this generates a 'Library' namespace friendly name for the input name "Te st"
cps::mapName {Te st} Concept Library
#Output:
#Te#20st
```

openExternal

Use openExternal to open an external URL in a new window.

Return Type

NONE

Syntax

```
cps::openExternal <url> <viewType> <itemType>
```

Parameters

Parameter	Type	Description
`url`	`STRING`	External URL to be opened . Eg. https://www.google.com This parameter is required.
`viewType`	`STRING`	Use the predefined value 'Text'. No other parameter value supported. This parameter is required.
`itemType`	`STRING`	Use the predefined value 'Text'. No other parameter value supported. This parameter is required.

Examples

```
#command to open google home page from SystemCapture.
cps::openExternal https://www.google.com Text Text
```

openHybridDock

This command displays a web page within a docked panel with the mentioned position, size, and dockArea. The position and size are applicable only if the docked widget is in a floating state.

On success, this command returns a non-negative integer representing the unique ID of the newly created tab.

Return Type

INT

Syntax

```
sdaUI::openHybridDock <title> <url> <dockName> ?isFloating? ?pos? ?size? ?dockArea?
```

Parameters

Parameter	Type	Description
`title`	`STRING`	Title of the docked panel. This parameter is required.
`url`	`STRING`	URL of the content to host. This parameter is required.
`dockName`	`STRING`	Name of the docked panel. This parameter is required.
`isFloating`	`BOOL`	Docked panel is floating or not. This parameter is optional. Default value is `false`.
`pos`	`LIST`	Position of the docked panel. This parameter is applicable only when docked is in floating state. This parameter is optional. Default value is `{0 0}`.
`size`	`LIST`	Size of the docked panel, width and height. This parameter is applicable only when docked panel is in floating state. This parameter is optional. Default value is `{800 800}`.
`dockArea`	`STRING`	Area of docked panel, valid values are ::cps::DA_LEFT,::cps::DA_RIGHT,::cps::DA_BOTTOM This parameter is optional. Default value is `::cps::DA_LEFT`.

Examples

```
# This will display the docked panel at the default position and size.
# It will not be in a floating state as the default value of isFloating argument is
false.
sdaUI::openHybridDock CADENCE www.cadence.com testtab02
# This command will return a unique ID
# 101

# This will display the docked panel at the specified position and size.
sdaUI::openHybridDock CADENCE www.cadence.com testtab02 true {100 100} {400 400} {}
# This command will return a unique ID
# 96
```

Related Commands

addDock

setDockWidgetUndockDefaultSize

openURLDialog

openURLTab

openHybridDockWithCloseHandler

This command displays a web page within a docked panel with a close handler

Return Type

INT

Syntax

```
sdaUI::openHybridDockWithCloseHandler <title> <url> <dockName> <close_handler> ?
isFloating? ?pos? ?size? ?dockArea?
```

Parameters

Parameter	Type	Description
`title`	`STRING`	Title of the docked panel This parameter is required.
`url`	`STRING`	Url of the content to host This parameter is required.
`dockName`	`STRING`	Name of the docked panel This parameter is required.
`close_handler`	`STRING`	Name of tcl procedure to receive a callback whenever a docked panel is closed by user or visibility of docked panel changes to hidden. Syntax of close handler should be Proc onclosedock {handle} { } This parameter is required.
`isFloating`	`BOOL`	Docked panel is floating or not. This parameter is optional.
`pos`	`INT`	Position of the docked panel. Note: position is only applicable when docked panel is in floating state. This parameter is optional. Default value is `{0 0}`.
`size`	`INT`	Size of the docked panel in width and height. Note: position is only applicable when docked panel is in floating state. This parameter is optional. Default value is `{800 800}`.
`dockArea`	`STRING`	Area of docked panel, valid values are ::cps::DA_LEFT,::cps::DA_RIGHT,::cps::DA_BOTTOM This parameter is optional. Default value is `::cps::DA_LEFT`.

Examples

```
# suppose user creates this function to be called when #openURLDialogWithCloseHandler
is called
proc onCloseDock { handle} {
}

# This will display a web page within a docked panel with a close handler
TCL&gt; sdaUI::openHybridDockWithCloseHandler CADENCE www.cadence.com testtab02
onCloseDock
```

openURLDialogWithCloseHandler

This command displays a web page within a dialog with a close handler.

Return Type

INT

Syntax

```
sdaUI::openURLDialogWithCloseHandler <title> <url> <name> <close_handler> ?icon? ?
position? ?size? ?isModal? ?isResizable?
```

Parameters

Parameter	Type	Description
`title`	`STRING`	Title of the tab This parameter is required.
`url`	`STRING`	Url of the content to host This parameter is required.
`name`	`STRING`	Name of the hybrid component. Use this name to look up the component handle using cps::findComponentByName This parameter is required.
`close_handler`	`STRING`	Name of tcl procedure to receive a callback whenever a docked panel is closed by user or visibility of docked panel changes to hidden. Syntax of close handler should be Proc onclosedock {handle} { } This parameter is required.

`icon`	`STRING`	Path to the dialog icon This parameter is optional. Default value is `{}`.
`position`	`INT`	Position of the web-view within the dialog. This is reserved for future use This parameter is optional. Default value is `{0 0}`.
`size`	`INT`	Size of the dialog in width and height This parameter is optional. Default value is `{800 800}`.
`isModal`	`BOOL`	Modality of the dialog. A value of 1 indicates a modal dialog This parameter is optional. Default value is `1`.
`isResizable`	`BOOL`	Resizability of the dialog This parameter is optional.

Examples

```
# suppose user creates this function to be called when #openURLDialogWithCloseHandler
is called
proc onCloseDock { handle} {
}

# This will display a web page within a docked panel with a close handler
sdaUI::openURLDialogWithCloseHandler CADENCE www.cadence.com testtab02 onCloseDock
```

readFile

Returns the contents of the file as a TCL string. Supports text format files only.

Return Type

STRING

Syntax

```
cpCommon::readFile <filePath>
```

Parameters

Parameter	Type	Description
`filePath`	`STRING`	Path of the text file which needs to be read. This parameter is required.

Examples

```
# This command displays the contents of the file below
set fileContents [cpCommon::readFile /home/jdoe/someasciitextfile]
puts $fileContents
```

resetWindowLayout

Reset window layout settings from the registry.

Return Type

void

Syntax

```
resetWindowLayout
```

Examples

```
cps::resetWindowLayout
```

selectWindow

Calls cps::selectWindow <windowName>

Return Type

BOOL

Syntax

```
selectWindow <windowName>
```

Parameters

Parameter	Type	Description
`<windowName>`	`STRING`	name of the window This parameter is required.

Examples

```
selectWindow window_name
```

setAutomationMode

Controls the automation mode. Enabling automation mode suppresses message box pop-ups.

To enable automation, set the mode as $cps::eCPS_AUTOMATION_ENABLED (numeric value 1)
To disable automation, set the mode to $cps::eCPS_AUTOMATION_DISABLED (numeric value of 2)

Mode $cps::eCPS_AUTOMATION_NOTSET (numeric value 0) is deprecated and should not be used.

Return Type

NONE

Syntax

```
cps::setAutomationMode <mode>
```

Parameters

Parameter	Type	Description
`mode`	`INT`	Allowed values - $cps::eCPS_AUTOMATION_ENABLED, $cps::eCPS_AUTOMATION_DISABLED, $cps::eCPS_AUTOMATION_NOTSET This parameter is required.

Examples

```
# store value of existing automation mode in a variable
set existingMode [cps::getAutomationMode]

# enable automation mode
cps::setAutomationMode $::cps::eCPS_AUTOMATION_ENABLED

# perform batch processing without UI interruptions
# end of batch processing

#reset automation mode
cps::setAutomationMode $existingMode
```

Related Commands

getAutomationMode

setCloseTCLHandler

Sets the Tcl procedure to be run when a dialog or docked panel is closed. You can set a Tcl procedure to receive a callback whenever a dialog or docked panel is closed or the visibility of a dialog box or docked panel changes to hidden.

Return Type

INT

Syntax

```
sdaUI::setCloseTCLHandler <handle> <Tcl_procedure>
```

Parameters

Parameter	Type	Description
`handle`	`STRING`	Handle of the dialog or docked panel This parameter is required.
`Tcl_procedure`	`STRING`	Name of Tcl procedure This parameter is required.

Examples

```
# on a dialog
set ret [sdaUI::hybridDialog TESTDATA www.cadence.com testcadence]
set dlgHandle [dict get $ret dialogHandle]
sdaUI::setCloseTCLHandler $dlgHandle "hybridCallback"

# on dock widget
set dockHandle [sdaUI::openHybridDock TESTDATA www.cadence.com sampleDock]
sdaUI::setCloseTCLHandler $dockHandle "hybridCallback"
```

Related Commands

sdaUI::hybridDialog

sdaUI::openHybridDock

setDockedWidgetVisibility

This Command sets the visibility of the dock widgets. This command works when dock widget is in pinned state.

Supported widget names for argument <dockWidgetName> are:
CP_TCL_WINDOW_DOCK - for Command Window
CP_PROPERTIES_DOCK - for Properties dock widget
CP_ERROR_VIOLATION_DOCK - for Violation Window
CP_DESIGN_DIFFERENCES_WINDOW - for Design Differences dock widget
CP_NAVIGATION_DOCK - for Navigation dock widget
CP_DESIGN_EXPLORER_DOCK - for Project Explorer dock widget
CP_ADD_COMPONENT_SEARCH_DOCK - for Search Window
CP_LOG_DOCK - for Session Log Window
CP_WORKFLOW_DOCK - for Workflow dock widget

Return Type

INT

Syntax

```
cps::setDockedWidgetVisibility <dockWidgetName> <state>
```

Parameters

Parameter	Type	Description
`dockWidgetName`	`STRING`	Specify the dock widget name. This parameter is required.
`state`	`BOOL`	Visibility state of Property Dock Widget This parameter is required.

Examples

```
1. Command to set the Property Dock Widget visible.
cps::setDockWidgetVisibility "CP_PROPERTIES_DOCK" true

2. Command to set the Property Dock Widget hide.
cps::setDockWidgetVisibility "CP_PROPERTIES_DOCK" false

3. If no argument given then it gives an error as "Wrong number of arguments
:cps::setDockWidgetVisibility strObjName strState argument 1".
cps::setDockWidgetVisibility

4. If one argument given then it gives an error as "Wrong number of arguments
:cps::setDockWidgetVisibility strObjName strState argument 2".
cps::setDockWidgetVisibility "CP_PROPERTIES_DOCK"
```

Related Commands

setDockedWidgetVisibilityOff

setPropertiesWidgetVisibility

setDockedWidgetVisibilityOff

Hides the specified docked widget, even if the widget is pinned.

The supported widget names for dockWidgetName are:
CP_TCL_WINDOW_DOCK - for Command Window
CP_PROPERTIES_DOCK - for Properties dock widget
CP_ERROR_VIOLATION_DOCK - for Violation Window
CP_DESIGN_DIFFERENCES_WINDOW - for Design Differences dock widget
CP_NAVIGATION_DOCK - for Navigation dock widget
CP_DESIGN_EXPLORER_DOCK - for Project Explorer dock widget
CP_ADD_COMPONENT_SEARCH_DOCK - for Search Window
CP_LOG_DOCK - for Session Log Window
CP_WORKFLOW_DOCK - for Workflow dock widget

Return Type

INT

Syntax

```
cps::setDockedWidgetVisibilityOff <dockWidgetName>
```

Parameters

Parameter	Type	Description
`dockWidgetName`	`STRING`	The name of the docked widget This parameter is required.

Examples

```
# This command sets the Property Dock Widget visibility off.
cps::setDockedWidgetVisibilityOff CP_PROPERTIES_DOCK
```

Related Commands

setDockedWidgetVisibility

setPropertiesWidgetVisibility

setPagePreviewerWidgetVisibility

Toggles page preview widget visibility

Return Type

BOOL

Syntax

```
cps::setPagePreviewerWidgetVisibility
```

setPropertiesWidgetVisibility

Shows the Properties docked widget on the right side. If the Properties dock widget is already visible, nothing happens.

Return Type

INT

Syntax

```
cps::setPropertiesWidgetVisibility
```

Examples

```
# To display Properties dock widget if already not visible
cps::setPropertiesWidgetVisibility
```

Related Commands

setDockedWidgetVisibility

setDockedWidgetVisibilityOff

setPulseDirectiveValue

Set the Pulse Directive Value for design attributes of currently opened design

Return Type

Status

Syntax

```
cps::setPulseDirectiveValue CUSTOMVAR <name> <value> string
```

Parameters

Parameter	Type	Description
`name`	`String`	name of design attribute to be set This parameter is required.

Examples

```
Tcl&gt; cps::setPulseDirectiveValue CUSTOMVAR ProgramName new string
1
Tcl&gt;

On Success returns 1

ProgramName is the name of design attribute
new is the value of design attribute
```

setWidgetSize

Changes the width and height of the specified widget.

Return Type

INT

Syntax

```
cps::setWidgetSize <widgetHandle> <width> <height>
```

Parameters

Parameter	Type	Description
`widgetHandle`	`INT`	Handle of the widget to be resized. This parameter is required.
`width`	`INT`	New width for the widget This parameter is required.
`height`	`INT`	New height for the widget This parameter is required.

Examples

```
# This command returns the dialog handle of the mentioned dialog name.
set widHandle [sdaUI::openURLDialog CADENCE www.cadence.com testtab02 {} {0 0} {1000
400} 1 1]
puts $widHandle
# 82

# This will set the width and height of the widget to 800x800
cps::setWidgetSize $widHandle 800 800
```

Related Commands

openURLDialog

showMessageDialog

Display a message box with buttons that the user can click. The return value depends on which button is clicked in the message box.
Clicking "OK" or "No", returns 0
Clicking"Yes", returns 1

In the message box, buttons are displayed according to severity (msgType) and buttonType1 and buttonType2.
For severity 0 , 1 , and 2, the possible combination of buttons are:
1. To show only "OK" button, set both buttons as blank(that is "") or set them as buttonType2 = "" and buttonType1 = "OK"
2. To show "Yes" and "No" buttons , set buttonType1 = "Yes" and buttonType2 = "No"
3. To show "Yes" and "Cancel" buttons, set buttonType1 = "Yes" and buttonType2 = "Cancel"

For severity 3, the only possible combination for buttons is:
1. To show "Yes", "No", and "Cancel" buttons, set buttonType1 = "Yes" and buttonType2 = "No". A third button "Cancel" is also displayed with this severity value.
2. To show only "Yes" and "Cancel" buttons, set buttonType1 = "Yes" and buttonType2 = "Cancel"

For severity 4, the only possible combination for buttons is:
1. To show "Yes" and "No" buttons, set buttonType1 = "Yes" and buttonType2 = "No"
2. To show "Yes" and "Cancel" buttons, set buttonType1 = "Yes" and buttonType2 = "Cancel"

Return Type

BOOL

Syntax

```
cps::showMessageDialog <title> <message> <msgType> <buttonType1> <buttonType2>
```

Parameters

Parameter	Type	Description
`title`	`STRING`	Title of the message box This parameter is required.
`message`	`STRING`	Message to be displayed in message box This parameter is required.
`msgType`	`INT`	Severity of the message, 0 = Error, 1= Warning, 2 = Information, 3 = 3 button dialog, 4 = 2 button dialog This parameter is required.
`buttonType1`	`STRING`	First button type. Possible values are "", "Yes" , and "OK" This parameter is required.
`buttonType2`	`STRING`	Second button type. Possible values are "", "No" , and "Cancel" This parameter is required.

Examples

```
# This example displays the application version with an OK button
cps::showMessageDialog version_info "version number is [cps::getAppVersion]" 2 "" ""

# With severity 0, 1, 2
# 1. To show only an OK button:
cps::showMessageDialog version_info "version number is [cps::getAppVersion]" 2 "" ""
# 2. To show "Yes" and "No" buttons
cps::showMessageDialog version_info "version number is [cps::getAppVersion]" 2 "Yes"
"No"
# 3. To show "Yes" and "Cancel" buttons
cps::showMessageDialog version_info "version number is [cps::getAppVersion]" 2 "Yes"
"Cancel"

# With severity 3
# 1. To show "Yes", "No", and "Cancel" buttons
cps::showMessageDialog version_info "version number is [cps::getAppVersion]" 3 "Yes"
"No"
# 2. To show only "Yes", and "Cancel" buttons
cps::showMessageDialog version_info "version number is [cps::getAppVersion]" 3 "Yes"
"Cancel"

# With severity 4
# 1. To show "Yes" and "No" buttons
cps::showMessageDialog version_info "version number is [cps::getAppVersion]" 4 "Yes"
"No"
# 2. To show "Yes" and "Cancel" buttons
cps::showMessageDialog version_info "version number is [cps::getAppVersion]" 4 "Yes"
"Cancel"
```

Related Commands

showMessageDialogWithCheckBoxBtn

showMessageDialogEx

An extended version for the cps::showMessageDialog and cps::showMessageDialogWithCheckBoxBtn commands depending upon the last parameter, it asks whether the user wants to show a message dialog with or without the "Do not show this message again" checkbox.

if it is passed true, a checkbox will be shown.
if it is passed false, a checkbox will not be shown. No default value

Return Type

INT

Syntax

```
showMessageDialogEx <title> <message> <msgType> <buttonType1> <buttontype2>
<pShowDialog>
```

Parameters

Parameter	Type	Description
`title`	`STRING`	Title of the message box This parameter is required.
`message`	`STRING`	Message to be displayed in message box This parameter is required.
`msgType`	`INT`	Severity of the message, 0 = Error, 1= Warning, 2 = Information, 3 = 3 button dialog, 4 = 2 button dialog This parameter is required.
`buttonType1`	`STRING`	First button type, its possible values are "", "yes" , "ok" This parameter is required.
`buttonType2`	`STRING`	Second button type, its possible values are "", "no" , "cancel" This parameter is required.
`pShowDialog`	`BOOL`	Argument to show or hide "Do not show this message again" checkbox from message dialog This parameter is required.

Examples

```
#with checkbox
showMessageDialogEx "Version Info" "version number is [cps::getAppVersion]" 2 "OK" "0"
true

#without checkbox
showMessageDialogEx "Version Info" "version number is [cps::getAppVersion]" 2 "OK" "0"
false
```

Related Commands

showMessageDialog

showMessageDialogWithCheckBoxBtn

showMessageDialogWithCheckBoxBtn

The message dialog will contain "Do not show this message again" checkbox, clicking it won't show a similar message next time for the same session.

Return value of function will depend on click of the button on the message box.
On click of button "OK" or "NO" will return 0
On click of button "Yes" will return 1

On the message box, buttons will be displayed according to severity (msgType) and buttonType1 and buttonType2.
With severity 0 , 1 , 2
Only possible combination of buttons are
1. "OK" when buttonType1 = "" , buttonType2 = "" or buttonType1 = "ok"
2. "yes", "no" , when buttonType1 = "yes" , buttonType2 = "no"
3. "yes", "cancel" , when buttonType1 = "yes" , buttonType2 = "cancel"

With severity 3
Only possible combination for buttons are
1. "yes", "no" "cancel" , when buttonType1 = "yes" , buttonType2 = "no" , a third button with "cancel" text will also be displayed with this severity value.
2. "yes", "cancel" , when buttonType1 = "yes" , buttonType2 = "cancel"

With severity 4
Only possible combination for buttons are
1. "yes", "no" , when buttonType1 = "yes" , buttonType2 = "no"
2. "yes", "cancel" , when buttonType1 = "yes" , buttonType2 = "cancel"

Return Type

BOOL

Syntax

```
cps::showMessageDialogWithCheckBoxBtn <title> <message> <messageType> <buttonType1>
<buttontype2>
```

Parameters

Parameter	Type	Description
`title`	`STRING`	Title of the message box This parameter is required.
`message`	`STRING`	Message to be displayed in message box This parameter is required.
`messageType`	`INT`	Severity of the message, 0 = Error, 1= Warning, 2 = Information, 3 = 3 button dialog, 4 = 2 button dialog This parameter is required.
`buttonType1`	`STRING`	First button type, its possible values are "", "yes" , "ok" This parameter is required.
`buttonType2`	`STRING`	Second button type, its possible values are "", "no" , "cancel" This parameter is required.

Examples

```
# This example creates a dialog box with a checkbox button, with the following title,
description and has two buttons Ok and Cancel
set title "Version Info"
set desc "version number is [cps::getAppVersion]"
cps::showMessageDialogWithCheckBoxBtn $title $desc 2 "Ok" "Cancel"
# Output will pop-up a message dialog with a checkbox button and above details
```

Related Commands

showMessageDialog

showMessageDialogEx

startTCPServer

Starts the specified TCP server. On successful server start, the registered server status handler is called with the status code of 1 and a message is displayed indicating success. After the server starts, clients connect to it using HTTP requests.

If the server fails to start, the handle is called with the status code of 0 and the reason for the failure is displayed.

Return Type

INT

Syntax

```
cps::startTCPServer <hostIP> <port>
```

Parameters

Parameter	Type	Description
`<hostIP>`	`STRING`	IP address of the current machine on which System Capture is running This parameter is required.
`<port>`	`INT`	Port number for the TCP server to listen for incoming requests This parameter is required.

Examples

```
# get the hostname
set hName [info hostname]
puts $hName

# This command starts the TCP server, 9001 is the port number where TCL server will
# listen for incoming requests
cps::startTCPServer $hName 9001
```

updateUserPreferences

Saves the changes in the User Preferences to system settings.

Return Type

INT

Syntax

```
cps::updateUserPreferences -autoSave {enable_disable_autoSave} -autoPan
{enable_disable_autoPan} -showGridLines {show_hide_gridLine} -showGridLocation
{show_grid_location} -recentProjectListCount {total_recent_projects} -theme
{theme_dark_light} -hiliteXnet {xnet_enable_disable} -checkOverlapNetName
{check_overlap_net_name} -autoSaveInterval {autosave_interval} -showProjectGridLines
{enable_disable_project_grid_lines:} -schematicSheetTheme {schematic_theme_dark_light}
-gridLinesColor {page_grid_lines_color} -schematicSheetBackgroundColor
{schematic_sheet_background_color}
```

Parameters

Parameter	Type	Description
`enable_disable_autoSave`	`INT`	Enables or disables autosave. 0 for disable and 1 for enable. This parameter is optional.
`enable_disable_autoPan`	`INT`	Enables or disables auto pan. 0 for disable and 1 for enable. This parameter is optional.
`show_hide_gridLine`	`INT`	Shows or hides gridlines. 0 to hide and 1 to show. This parameter is optional.

`show_grid_location`	`INT`	Shows or hides the grid location. 0 to hide and 1 to show. This parameter is optional.
`total_recent_projects`	`INT`	Number of recent projects to be shown in the recent projects list This parameter is optional.
`theme_dark_light`	`STRING`	Changes the application theme. Use "dark" or "light". This parameter is optional.
`xnet_enable_disable`	`INT`	Enables or disables xnets This parameter is optional.
`check_overlap_net_name`	`INT`	Enables or disables the overlapping net name check This parameter is optional.
`autosave_interval`	`INT`	Sets the auto save interval duration This parameter is optional.
`enable_disable_project_grid_lines`	`INT`	Enables or disables the project gridlines This parameter is optional.
`schematic_theme_dark_light`	`STRING`	Changes the schematic editor theme. Use "dark" or "light". This parameter is optional.
`page_grid_lines_color`	`STRING`	Changes the grid lines color. This parameter is optional.
`schematic_sheet_background_color`	`STRING`	Changes the background color This parameter is optional.

Examples

```
updateUserPreferences -autoSave 0 -autoPan 1 -showGridLines 1 -showGridLocation 0 -
recentProjectListCount 5 -theme dark -hiliteXnet 0 -checkOverlapNetName 0 -
autoSaveInterval 1 -showProjectGridLines 0 -schematicSheetTheme light -gridLinesColor
#0000ff -schematicSheetBackgroundColor Mid
#Passing an invalid color code option for gridLinesColor will set the system default
#gridline color for the chosen schematic theme
```

viewBoardFromProjectViewer

Shows board from the project viewer

Return Type

BOOL

Syntax

```
viewBoardFromProjectViewer
```

Examples

```
viewBoardFromProjectViewer
```

14

Internal Commands

Documentation is not available for internal commands.

- `CPBF::addUsers`
- `CPBF::canShowDesignList`
- `CPBF::changeLockingMode`
- `CPBF::commit`
- `CPBF::displayErrorMessage`
- `CPBF::downloadFile`
- `CPBF::dumpDesignList`
- `CPBF::dumpInFile`
- `CPBF::dumpObjectInfo`
- `CPBF::dumpOperationStatus`
- `CPBF::dumpPendingNetlists`
- `CPBF::dumpPulseBrowseUI`
- `CPBF::dumpPulseHealthReport`
- `CPBF::dumpSegShareUI`
- `CPBF::dumpShareUI`
- `CPBF::dumpTooltip`
- `CPBF::dumpVersionGraph`
- `CPBF::getConfig`

- `CPBF::getDesignID`
- `CPBF::getLatestObjectID`
- `CPBF::getMasterID`
- `CPBF::getNetlistForUrl`
- `CPBF::getPendingUpdates`
- `CPBF::getServerDataValue`
- `CPBF::getUserInfo`
- `CPBF::join`
- `CPBF::manageDesign`
- `CPBF::managePulse`
- `CPBF::openPulseHome`
- `CPBF::processEvents`
- `CPBF::refreshPermissions`
- `CPBF::reimportNetlist`
- `CPBF::removeProject`
- `CPBF::removeUser`
- `CPBF::restoreProject`
- `CPBF::revert`
- `CPBF::rollback`
- `CPBF::rollbackObjects`
- `CPBF::rollbackToVersion`
- `CPBF::setConfig`
- `CPBF::setVersioning`
- `CPBF::share`
- `CPBF::showPulseHealthReport`

- `CPBF::showPulseOpStatus`
- `CPBF::unlock`
- `CPBF::updateDesign`
- `CPBF::updateDesignInternal`
- `CPBF::updateUser`
- `CPBF::versionGraph`
- `CPBF::waitForPulseOpInProgress`
- `addActionToToolById`
- `addActionToToolItem`
- `addComponentBridge`
- `addComponentX`
- `addConnection`
- `addMenuToContextMenu`
- `addToolItemToToolBarById`
- `addToolItemToToolBarById`
- `annotate`
- `applylabel`
- `autoDrawWire`
- `bindBlockDataFromProjectViewer`
- `bindBoardFromProjectViewer`
- `captureLibraries`
- `changeRoot`
- `checkConnectivity`
- `compareCheckSumAndInvokeVDDForBlock`
- `cpSchT::getBaseURL`

- `cpSchT::setGlobalHybridBrowserDlgHandle`
- `cpSchT::setGlobalHybridBrowserHandle`
- `cpSchT::setGlobalHybridRepBrowserDlgHandle`
- `cpSchT::setGlobalHybridRepBrowserHandle`
- `cpSchT::setUnifiedSearchLoginStatus`
- `cpSchT::unifiedSearchOpenDetailsPanel`
- `cpSchT::unifiedSearchPlacePart`
- `cpb::appendToLog`
- `cpb::checkoutProjectViewOptions`
- `cpb::checkoutSymbolView`
- `cpb::destroyUnifiedSearch`
- `cpb::destroyWorkFlow`
- `cpb::downloadFile`
- `cpb::dumpImportPhysicalDifferences`
- `cpb::exportDesignCache`
- `cpb::exportLst`
- `cpb::getURLParams`
- `cpb::hasPackagingErrors`
- `cpb::isUnicornEnabled`
- `cpb::loadUnifiedSearchUrls`
- `cpb::replaceVariants`
- `cpb::setGlobalHybridBrowserDlgHandle`
- `cpb::setGlobalHybridBrowserHandle`
- `cpb::setUnifiedSearchLoginStatus`
- `cpb::unifiedSearchAdd`

- `cpb::unifiedSearchOpenDetailsPanel`
- `cpfm::getMasterId`
- `cpfm::getServerDataValue`
- `cpfm::getVersionId`
- `cpfm::invokeTclMethod`
- `cpfm::operationFinished`
- `cppt::ascendToParentView`
- `cppt::cleanWidgetManagerStores`
- `cppt::collapseAllJunction`
- `cppt::descendVrmOutPin`
- `cppt::dumpOpenedPowerTreeView`
- `cppt::expandAllJunction`
- `cppt::selectObjectInPowerTree`
- `cppt::selectObjectViaMouseClick`
- `cppt::toggleAllJunction`
- `cppt::toggleJunction`
- `cppt::toggleView`
- `cppt::zoomIn`
- `cppt::zoomOut`
- `cps::addActionToMenu`
- `cps::addActionToTool`
- `cps::addBlockToDesign`
- `cps::addLib`
- `cps::addMenuToMenu`
- `cps::addMenuToMenu`

- `cps::addSection`
- `cps::addSeparatorToContextMenu`
- `cps::addSeparatorToToolBar`
- `cps::addToContext`
- `cps::addToDock`
- `cps::addToTab`
- `cps::addToolItemToToolBar`
- `cps::addWidgetToStackInDock`
- `cps::askToSave`
- `cps::captureScreenShot`
- `cps::changeStartPageState`
- `cps::checkIfSyscapUpdateAvailable`
- `cps::closeView`
- `cps::collapseAll`
- `cps::collapseThisGroup`
- `cps::collapseTree`
- `cps::connectDEToolBarButtonToDEDock`
- `cps::connectDEToolBarButtonToPagePreviewerDock`
- `cps::createAction`
- `cps::createBrowserDialog`
- `cps::createComponent`
- `cps::createDesignExplorer`
- `cps::createDialog`
- `cps::createDock`
- `cps::createFloatingWidget`

- `cps::createHierView`
- `cps::createItem`
- `cps::createMultiTableView`
- `cps::createMultiTreeView`
- `cps::createNewBlock`
- `cps::createProgressBar`
- `cps::createToolItem`
- `cps::createTreeView`
- `cps::createWidget`
- `cps::deleteEmptyWorkSpace`
- `cps::deleteItem`
- `cps::deleteItems`
- `cps::deleteLib`
- `cps::deleteMenuFromMenuBar`
- `cps::deleteMenuItem`
- `cps::deleteProgressBar`
- `cps::deleteSeparatorFromMenu`
- `cps::deleteToolItem`
- `cps::deleteTreeNode`
- `cps::dumpViolations`
- `cps::enableDebugging`
- `cps::enableDisableEvents`
- `cps::exitApp`
- `cps::expandAll`
- `cps::expandThisGroup`

- `cps::expandTree`
- `cps::find`
- `cps::findComponentByName`
- `cps::findWidget`
- `cps::generateHierSymbol`
- `cps::getBridgeId`
- `cps::getComponentName`
- `cps::getCurrentVisibleDockWidget`
- `cps::getDefaultItemName`
- `cps::getDefaultItemType`
- `cps::getDefaultViewType`
- `cps::getErrorString`
- `cps::getHierViewData`
- `cps::getHierarchyExplorerData`
- `cps::getMPSSession`
- `cps::getMenuId`
- `cps::getProjDir`
- `cps::getProjectExplorerData`
- `cps::getRecentProjectPaths`
- `cps::getRecentProjectThumbnails`
- `cps::getTCPServerRequestHandler`
- `cps::getTCPServerStatusHandler`
- `cps::getTempDirectory`
- `cps::getTree`
- `cps::getTreeNodeChildren`

- `cps::getTreeNodeDisplayString`
- `cps::getTreeNodeIcon`
- `cps::getTreeNodeItemType`
- `cps::getTreeNodeMenuID`
- `cps::getTreeNodeOpenCommand`
- `cps::getTreeNodePtrByName`
- `cps::getTreeNodeViewType`
- `cps::getTreeRootData`
- `cps::getWidgetPtr`
- `cps::hidePageWidgetDock`
- `cps::highlightDETabOnSelection`
- `cps::isDesignDirty`
- `cps::isLogLevelSet`
- `cps::isMenuPresent`
- `cps::isPSpiceEnabled`
- `cps::isRegisteredCommand`
- `cps::isTerminated`
- `cps::issueCommand`
- `cps::issueWidgetCommand`
- `cps::launchAbout`
- `cps::makeWebPageTransparent`
- `cps::makeWidgetTransparent`
- `cps::registerMPSScriptCallback`
- `cps::registerTCPServerRequestHandler`
- `cps::registerTCPServerStatusHandler`

- `cps::sendMPS`
- `cps::setAutoShapesWidgetVisibility`
- `cps::setCentralWidget`
- `cps::setComponentName`
- `cps::setConnectivityWidgetVisibility`
- `cps::setContraintManagerVisibility`
- `cps::setCurrentItemInTree`
- `cps::setDefaultItemName`
- `cps::setDefaultItemType`
- `cps::setDefaultViewType`
- `cps::setDesignDifferencesWindowVisibility`
- `cps::setDesignExplorerCurrentWindow`
- `cps::setDialogContent`
- `cps::setDialogParent`
- `cps::setDialogSettings`
- `cps::setDockWidgetIcon`
- `cps::setEditPropertiesCurrentWindow`
- `cps::setExplorerWidgetVisibility`
- `cps::setFormatWidgetVisibility`
- `cps::setHierViewData`
- `cps::setInsertAfterPagesCommandOnItem`
- `cps::setInsertBeforePagesCommandOnItem`
- `cps::setInterpContext`
- `cps::setLogLevel`
- `cps::setPageWidgetDockTitle`

- `cps::setPageWidgetFloating`
- `cps::setPageWidgetPinned`
- `cps::setPcbPartitionWidgetVisibility`
- `cps::setPhysicalEditorWidgetVisibility`
- `cps::setProperty`
- `cps::setSelectionFilterVisibility`
- `cps::setSize`
- `cps::setSpecialBodiesWidgetVisibility`
- `cps::setSuppressDialog`
- `cps::setTheme`
- `cps::setWidgetName`
- `cps::setWindowMode`
- `cps::setWorkspaceMode`
- `cps::showDockWidget`
- `cps::showHelp`
- `cps::showHidePageWidget`
- `cps::showTextInput`
- `cps::showTutorials`
- `cps::startComponent`
- `cps::tabDataDump`
- `cps::undo`
- `cps::unsetLogLevel`
- `cps::updateDockVisibilityAsPerPagePreviewer`
- `cps::updateHierarchyExplorer`
- `cps::updateItemPosToPageWidget`

- `cps::updatePagePreviewerStatus`
- `cps::updateProgressBarStatus`
- `cps::updateProjectExplorer`
- `cps::updateTreeNode`
- `cps::updateViewTitle`
- `createAction`
- `createItem`
- `createToolItem`
- `crefValidator`
- `definePinArrowType`
- `deleteBlock`
- `deleteContextMenuItem`
- `deleteItems`
- `deleteLib`
- `deleteSchPage`
- `discardSchPage`
- `display`
- `dm`
- `dockSchPage`
- `drawMultiWire`
- `dump`
- `dumpGrid`
- `dumpNetData`
- `dumpToWindow`
- `dumpVariantTable`

- `editBypass`
- `editPropsFromSPath`
- `exportBlockPhysicalData`
- `extractEcSet`
- `extractdata`
- `fetchCommandArgsAsVec`
- `filterVariantTable`
- `findObjConstraint`
- `followRoute`
- `generateEndToEndConnectivityReport`
- `generateHierarchicalSymbol`
- `generatePCBPartition`
- `generateSymbolfromSystemBlock`
- `generateSystemDesignReport`
- `getTclEnv`
- `hidePhysicalNetNames`
- `hierarchyDump`
- `highlight`
- `importCaptureDesign`
- `importConstraintFile`
- `importDEHDLBlock`
- `importDEHDLDesign`
- `insertSchPage`
- `launchBrowser`
- `launchUserPreferences`

- `launchVariantTable`
- `manualUpdateComponentPM`
- `manualupdateparts`
- `modifyActionShortcut`
- `modifyTable`
- `moveSchPage`
- `moveVariantTableColumn`
- `moveWire`
- `navigate`
- `ngUpdate`
- `openAuditLog`
- `openBoardFile`
- `openPagePost`
- `openProjectCore`
- `openSymbol`
- `operateSplitter`
- `packageOptions`
- `placeComponent`
- `placeTermination`
- `postExportPhysical`
- `postNavigate`
- `postSelectObject`
- `preAddComponent`
- `preEditPropsPathTool`
- `preExportPhysical`

- `preSelectObject`
- `preferenceGeneral`
- `printArray`
- `printFileAttributes`
- `projectDump`
- `projectPreferences`
- `reImportExternalAllegroBoard`
- `refreshDesignExplorer`
- `refreshPageWidget`
- `relaunchBrowser`
- `removeCommandsNeedNotToDocument`
- `removeFromXNet`
- `removeLibs`
- `removeLock`
- `renameItem`
- `renameObject`
- `renameSchPage`
- `reopenProject`
- `reorderItems`
- `replaceVariants`
- `reportSystemDesignViolations`
- `resolveBlockDesDiff`
- `resolveError`
- `resolveSystemDesignErrorViolation`
- `restoreVariantDB`

- `runPowerDC`
- `runSymbolScript`
- `saveSchPage`
- `sch::addPropNameVal`
- `sch::bindBlockData`
- `sch::changeTocNumberingMode`
- `sch::compareCheckSumForBlockVDD`
- `sch::createBlock`
- `sch::crefGoToPage`
- `sch::dbGetPinDataStringValue`
- `sch::deleteProperties`
- `sch::disablePartStatusQuery`
- `sch::dumpGeneralProperty`
- `sch::dumpProjectProperty`
- `sch::dumpSchLevelNetInstData`
- `sch::enablePartStatusQuery`
- `sch::exportBlockPhysicalData`
- `sch::exportPCB`
- `sch::exportPCBPartition`
- `sch::generateAliasMismatchReport`
- `sch::generateEndToEndConnectivityReport`
- `sch::generateHierarchicalSymbol`
- `sch::generateSchLevelCrefData`
- `sch::generateSymbolFromData`
- `sch::generateSymbolfromSystemBlock`

- `sch::generateSystemDesignDebugReport`
- `sch::getAttachment`
- `sch::getCheckOutBy`
- `sch::getDebugOption`
- `sch::getHierNameFromSpath`
- `sch::getModifiedBy`
- `sch::getObjectState`
- `sch::getOrCADEncodedBitMapData`
- `sch::getPSpiceSimulationTimerState`
- `sch::getSharedAreaVersion`
- `sch::getTDAComments`
- `sch::getTDAtoolTip`
- `sch::getWorkAreaVersion`
- `sch::highlightCanvasSelectionInVeditor`
- `sch::importElectricalCsetsFileDialog`
- `sch::importPinDelayFileDialog`
- `sch::importTechnologyFileDialog`
- `sch::insertItem`
- `sch::isObjectCheckedOut`
- `sch::lockComponent`
- `sch::markViolationMsgFixed`
- `sch::parseCsvData`
- `sch::reImportBlock`
- `sch::reImportExternalAllegroBoard`
- `sch::reImportExternalAllegroBoardFromProjectViewer`

- `sch::readDebugOption`
- `sch::refreshCell`
- `sch::refreshDockedCM`
- `sch::refreshSymbol`
- `sch::regenerateSchLevelCrefData`
- `sch::registerScriptCallback`
- `sch::renamePage`
- `sch::reportSystemDesignViolations`
- `sch::resolveError`
- `sch::resolveSystemDesignErrorViolation`
- `sch::resolveVDDLoadBlocksDiff`
- `sch::runDesignRule`
- `sch::runDesignRules`
- `sch::saveBlock`
- `sch::sdaFixDb`
- `sch::selectBlockInstance`
- `sch::selectCRefs`
- `sch::selectionFilterShortcut`
- `sch::setDRC`
- `sch::setDebugOption`
- `sch::setDesignRoot`
- `sch::setParsedCsvData`
- `sch::showHideCRefsOnCanvasFromAction`
- `sch::showHidePhysicalNetNames`
- `sch::showHidePhysicalNetNamesFromAction`

- `sch::startPSpiceSimulationTimer`
- `sch::statsReporting`
- `sch::stopPSpiceSimulationTimer`
- `sch::unregisterScriptCallback`
- `sch::updateBlocksAfterLoadBlocksDiff`
- `sch::updateCustomVariables`
- `sch::updateNetVoltageForSystemDesign`
- `sch::updateNoXNetProperty`
- `sch::updatePSpiceToolTipAndProfileName`
- `sch::updatePageManipulationPageList`
- `sch::updatePasteRepeatForSession`
- `sch::uprevSystemDesignData`
- `sdaReliability::GetDirectiveValue`
- `sdaReliability::areAdvancedAuditRulesAvailable`
- `sdaReliability::crossProbeViolatingObject`
- `sdaReliability::crossprobeComponent`
- `sdaReliability::crossprobeSubcircuit`
- `sdaReliability::findSubCircuitsMatch`
- `sdaReliability::generateConnectionMatrix`
- `sdaReliability::getInstancePropAliasValue`
- `sdaReliability::getLocale`
- `sdaReliability::getNormalisedValue`
- `sdaReliability::get_devicetype_subtype`
- `sdaReliability::listSchematicAuditCategories`
- `sdaReliability::listSchematicAuditRules`

- `sdaReliability::openAuditDashboard`
- `sdaReliability::registerSchematicAuditRule`
- `sdaReliability::runSchematicAuditOnCategory`
- `sdaReliability::setAuditRules`
- `sdaReliability::setReliabilityDirective`
- `sdaReliability::setWaiveFlag`
- `sdaReliability::updateProgressBarforRules`
- `sdaUI::hybridTab`
- `sdaUI::showHybridTab`
- `searchText`
- `selectSchPage`
- `selectVariantTable`
- `setDRC`
- `setInsertAfterPagesCommandOnItem`
- `setPageWidgetDockTitle`
- `setPageWidgetFloating`
- `setPageWidgetPinned`
- `setReplaceBrowserDlgParams`
- `setReplaceBrowserDlgParent`
- `setReplaceBrowserDlgVisibiltiy`
- `setTextMargin`
- `showBlockages`
- `showDockWidget`
- `showFloatingBrowser`
- `showHidePageWidget`

- `showReplaceFloatingBrowser`
- `sort`
- `sortVariantTable`
- `unselectObject`
- `updateComponent`
- `updateConnectivity`
- `updateDesignExplorer`
- `updateNetGroupMembers`
- `updateNoXNetProperty`
- `updatePCBPartition`
- `updatePinResolveStatus`
- `updateRootDesign`
- `viewBoardFile`
- `workflowExec`
- `workflow::executeCommandExec`
- `workflow::executeCommandExecWait`
- `workflow::generateLayoutReport`
- `workflow::generateLayoutReviewReports`
- `workflow::getMPSArg`
- `workflow::getSummaryReportPath`
- `workflow::getTempDirPath`
- `workflow::isWindowsOS`
- `workflow::launchPFM`
- `workflow::launchSyscap`
- `workflow::openAllegroViewerPlus`

- `workflow::openCommandPrompt`
- `workflow::openFileExplorer`
- `workflow::openOutputFolderInCommandPrompt`
- `workflow::openOutputFolderInFileExplorer`
- `workflow::openPCBEditorWithSRM`
- `workflow::openPhysicalViewInCommandPrompt`
- `workflow::openPhysicalViewInFileExplorer`
- `workflow::openweb`
- `workflow::printDebug`
- `workflow::printError`
- `workflow::replaceDirectiveValuesWithSubstitution`
- `workflow::reviewLayoutInSyscap`