cadence®

Virtuoso Parameterized Cell SKILL Reference

Product Version IC23.1
August 2023

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

Contents

3
Express Pcells Data Management Functions 169

1

Parameterized Cell Functions

This topic provides information about creating parameterized cells (Pcells) using Cadence® SKILL language and the Pcell graphical user interface (GUI). It describes the safety rules for creating SKILL Pcells, information about Pcell master cells and submaster cells, and descriptions of the supported `pc` functions. This user guide describes how to create graphical and SKILL Pcells.

Important

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

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

- The Virtuoso Studio design environment and application infrastructure mechanisms designed to support consistent operations between all Cadence® tools.
- The applications used to design and develop integrated circuits in the Virtuoso Studio design environment, notably, the Virtuoso Layout Suite, and Virtuoso Schematic Editor.
- The Virtuoso Studio design environment technology file.

Most of the supported `pc` SKILL functions correspond to the commands available only through the graphical user interface, and they can only be used to create Pcells for the graphical Pcell environment. However, there are four `pc` Pcell functions that you can use within the body of SKILL Pcell code:

- pcExprToString
- pcFix
- pcRound
- pcTechFile

Cadence recommends using Virtuoso® relative object design (ROD) functions in your Pcell code. ROD is a high-level language for defining layout connectivity, geometries, and the relationships between them.

Related Topics

Safety Rules for Creating SKILL Pcells

Recommended, Supported SKILL Functions for Pcells

Recommended and Supported SKILL++ Functions for Pcells

Use of SKILL Procedures from within Pcells

Macros in Pcell SKILL Code

Physical Limits for Functions

Using the SKILL Operator ~> with Pcells

What Is a Parameterized Cell?

Relative Object Design Concepts

pcExprToString

```
pcExprToString(
     g_ilExpr
     )
     => t_string
```

Description

Converts a SKILL expression to a string. The Pcell compiler uses this function to create labels that display the value of an expression as a string enclosed in quotes.

Arguments

g_ilExpr	SKILL expression you want to convert to a string.

Value Returned

t_string	The string equivalent of the SKILL expression, enclosed in quotes. If the value of the expression is `nil` or an error occurred, an empty string is returned.

Examples

```
pcExprToString( techGetSpacingRule( tfId "minWidth" "metal1" ) )
```

System retrieves the value of the minimum width rule for the `metal1` layer from the technology file and then converts that value to a string enclosed in quotes.

Related Topics

pcFix

pcRound

pcTechFile

Recommended, Supported SKILL Functions for Pcells

pcTechFile

```
pcTechFile(
     g_expression
     )
     => g_result
```

Description

Evaluates an expression contained in a string. The Pcell compiler uses this function as an envelope around stretch expressions that access information from a technology file. This function prevents any symbols used in the technology file access expression from being defined as parameters of the Pcell.

Arguments

g_expression	The expression you want the system to evaluate when the Pcell containing the expression is evaluated.

Value Returned

g_result	The value that resulted from evaluating the expression. If an error occurs, look in the CIW or at the `CDS.log` file for a message about the error.

Examples

```
pcTechFile( llp~>minWidth )
```

The symbol `minWidth` is used only during evaluation of the expression; it is not treated as a Pcell parameter. Here `llp` is an internal variable, `~>` is an operator, and `minWidth` is recognized by the system as a symbol.

If you did not use the `pcTechFile` function, the system would assume that `minWidth` is a Pcell parameter.

Related Topics

pcFix

pcRound

pcExprToString

2

Graphical Parameterized Cell Functions

This topic describes each of the supported `pc` functions you can use to create Pcells in the graphical Pcell environment. Do not use the functions in this section within the body of a SKILL Pcell code.

Graphical Pcell Functions

This section provides syntax, descriptions, and examples for graphical Pcell functions.

Related Topics

Creating Graphical Parameterized Cells

auHiUltraPCell

```
auHiUltraPCell(
     [ t_filename ]
     )
     => t / nil
```

Description

Displays the Ultra Pcell form to let you create an Ultra Pcell by compiling multiple Pcells into one cell. Optionally, you can save the Ultra Pcell SKILL code in a file by specifying the *t_filename* argument. For a description of Ultra Pcells, see Make Ultra Pcell Command

Ultra Pcells have the following requirements:

- Multiple Pcell layouts need not be in the same cell, although they must be in the same library as the completed ultra Pcell.
- Each Pcell must compile successfully.
- Each Pcell must have identical parameters and default values. No Pcell can have more or fewer or different parameters from the other Pcells. When the Ultra Pcell is compiled, it has all of the parameters, which work as designed in each Pcell.
- The selector parameter must be unique. It cannot match any other parameter.
- *type*, *objType*, *name*, *cell*, and *status* are reserved words and cannot be used as selector parameters.

Arguments

t_filename	Filename (or path and filename) in which you want to save the SKILL code output by the *Make Ultra Pcell* command.

Value Returned

t	The form opened, and optionally, the SKILL file was created.
nil	The form did not opened or the SKILL file was not created;.

Examples

```
auHiUltraPCell( "myUltraPcell.il" )
```

Creates an Ultra Pcell, saving the Pcell code in the file named `myUltraPcell.il`.

pcColinearPoints

```
pcColinearPoints(
     l_pointList1
     l_pointList2
     l_pointList3
     )
     => t / nil
```

Description

Verifies whether the three coordinates specified are collinear lying on or passing through the same straight line and orthogonal parallel to the X or Y axis.

Arguments

l_pointList1	First list of coordinates, specified as a list of two floating point numbers in the following format: `'(x y)`
l_pointList2	Second list of coordinates, specified as a list of two floating point numbers in the following format: `'(x y)`
l_pointList3	Third list of coordinates, specified as a list of two floating point numbers in the following format: `'(x y)`

Value Returned

`t`	The three specified coordinates are collinear and orthogonal.
`nil`	The three specified coordinates are not collinear and orthogonal.

Examples

```
pcColinearPoints( '( 1.0 2.0 ) '( 1.3 2.0 ) '( 3.0 2.0 ) )
```

Verifies whether the three coordinates listed are collinear and orthogonal.

pcConcatOrient

```
pcConcatOrient(
     t_orientation1
     t_orientation2
     )
     => t_concatOrient / R0
```

Description

Concatenates two specified object orientations into one.

Arguments

t_orientation1	String specifying the first orientation.
	`Valid Values: "R0", "R90", "R180", "R270", "MX", "MY", "MR90", "MR270"`
	`MR90 is equivalent to MXR90`
	`MR270 is equivalent to MYR90`

t_orientation2	String specifying the second orientation.
	`Valid Values: "R0", "R90", "R180", "R270", "MX", "MY", "MR90", "MR270"`
	`MR90 is equivalent to MXR90`
	`MR270 is equivalent to MYR90`

Value Returned

t_concatOrient	Orientation resulting from the concatenation.
`"R0"`	Returns `"R0"` if the concatenation fails.

Examples

```
pcConcatOrient("R0" "R90") => "R90"
```

Concatenates the orientation `R0` with `R90`, resulting in an orientation of `R90`.

pcDefineCondition

```
pcDefineCondition(
     d_cvId
     l_figs
     l_namelist
     g_condition
     g_stretch
     f_adjust
     )
     => d_condId / nil
```

Description

Specifies that the conditional inclusion of a list of objects is controlled by a given SKILL expression. Also specifies the inclusion of a dependent stretch control line. If the SKILL expression evaluates to a value other than `nil`, the objects are included.

Arguments

d_cvId	The database ID of the cellview in which the conditional inclusion is defined.
l_figs	List of objects controlled by the same conditional expression.
l_namelist	List of symbols referenced in the conditional expression. These become parameters of the Pcell.
g_condition	SKILL expression controlling the inclusion of the selected objects in the instance.
g_stretch	SKILL symbol specifying the name of a dependent stretch control line associated with this conditional inclusion. Use `nil` if there is no dependent stretch control line.
f_adjust	Amount by which a dependent stretch is adjusted from its reference dimension if *g_condition* evaluates to `nil`. This argument is ignored if *g_stretch* is `nil`.

Value Returned

d_condId	Group ID used to store the details of the conditional inclusion.
`nil`	Returned if *d_cvId* does not identify a cellview or *l_figs* does not contain any objects.

Examples

```
pcDefineCondition(cv figs list('Q_bar 'Reset) "Q_bar && Reset" "condStretch" 12.25)
```

Defines a conditional expression `Q_bar && Reset` in cellview *`d_cvId`* that controls the inclusion of the objects in *`l_figs`*. It declares `Q_bar` and `Reset` as symbols used as parameters for the Pcell. It also defines the stretch control line `condStretch` as dependent on this conditional inclusion, with the stretch value decreased 12.25 from the reference dimension if `Q_bar && Reset` evaluates to `nil`.

pcDefineInheritParam

```
pcDefineInheritParam(
     d_instId
     s_parameter
     g_value
     l_namelist
     )
     => d_inheritGroup / nil
```

Description

Specifies that a parameter of an instance of a Pcell takes its value from a parameter definition of the enclosing cellview. The selection filter allows you to select only instances.

Arguments

d_instId	Database ID of an instance of a Pcell.
s_parameter	Name of a parameter in the master cellview of *d_instID*.
g_value	SKILL expression for the value of *s_parameter* in *d_instID*.
l_namelist	List of symbols used in *g_value*. These become parameters of the Pcell.

Value Returned

d_inheritGroup	Group ID used to record the details of the inherited parameter.
nil	Returned if *d_instID* does not identify an instance or if there are no inherited parameters defined in it.

Examples

```
nfetWidthInheritance = pcDefineInheritParam(nfet1 'w 'nfetWidth list('nfetWidth))
```

Causes parameter `w` on instance `nfet1` to take its value from the `nfetWidth` parameter of the enclosing cellview. It declares `nfetWidth` as a parameter of the enclosing cellview.

pcDefineParamCell

```
pcDefineParamCell(
     d_cellviewId
     [ 'disablePrompt ]
     )
     => t / nil
```

Description

Allows you to compile a graphical Pcell (super) master in the database from the cellview you specify. If you do not compile a Pcell before you place an instance of it in another design, the system interprets the design as a standard fixed cell instead of a Pcell. Each time you edit a graphical Pcell, you must recompile it so that all placed instances reflect the changes.

You must have already defined parameters for the cellview.

Arguments

l_cellviewId	Database ID for the cellview (cellview ID). This argument is required.
'*disablePrompt*	If set to `t`, a prompt, which is displayed during the Pcell master compilation, gets disabled whenever no parameter is specified in the cell.

Value Returned

`t`	Parameterized (super) master was created.
`nil`	Parameterized (super) master was not created.

Examples

```
pcDefineParamCell( cellviewId )
```

Compiles a (super) master cell from the cellview identified by the variable *cellviewId*.

pcDefineParamLabel

```
pcDefineParamLabel(
     d_labelId
     S_height
     )
     => t / nil
```

Description

Defines a parameterized label. A parameterized label is not an instance name, but a label displaying values within a Pcell, such as width and height of gates. This function should be used only when creating graphical Pcells.

Arguments

d_labelId	Database ID for the label you want to parameterize. The label is a SKILL expression evaluated in the instance.
S_height	Height of the text in user units. Valid Values: any string or symbol

Value Returned

t	Returned if the label is converted to a parameterized label.
nil	Returned if the label is not converted to a parameterized label.

Examples

```
labelId=dbCreateLabel(geGetEditCellView() "poly1"
20:10 "width*height" "lowerLeft" "R0" "stick" 2)
pcDefineParamLabel(labelId "2")
```

The stretch parameters `width` and `height` are defined before using the example. Creates a label using the values of `width` times `height`. In this example, database ID produced by the `dbCreateLabel` function is assigned to a variable named `labelId`. Defines *d_labelId* as a parameterized label whose text is evaluated when you place the Pcell. The height of the label is 2 microns.

pcDefineParamLayer

```
pcDefineParamLayer(
     d_cvId
     l_shapes
     g_layerExpr
     l_namelist
     [ g_purposeExpr ]
     )
     => d_paramLayerId
```

Description

Specifies that a set of shapes has its layer and purpose determined by a parameter definition in the cellview. The selection filter prevents you from selecting instances or stretch control lines.

Arguments

d_cvId	The database ID of the cellview in which the parameter applies.
l_shapes	List of shapes in *d_cvId* that take their layer from a parameter of the cell.
g_layerExpr	SKILL expression for the layer to which the specified shapes belong. This expression evaluates to a string that must be a valid layer name.
l_namelist	List of symbols referenced in *g_layerExpr*. These become parameters of the cell.
g_purposeExpr	SKILL expression for the layer purpose to which the specified shapes belong. This expression evaluates to a string that must be a valid layer purpose.

Value Returned

d_paramLayerId	Group ID used to record the details of the layer parameter.

Examples

```
diffLayerParam =
pcDefineParamLayer(geGetEditCellView(w=getCurrentWindow( ))
geGetSelSet(w) 'diffLayer list("nDiff") list('diffLayer))
```

Causes the layer of the set of selected shapes to be determined by the value of the parameter `diffLayer`. It specifies `nDiff` as the default value for `diffLayer` and declares `diffLayer` as a parameter of the cell.

pcDefineParamPath

```
pcDefineParamPath(
     d_pathId
     S_param
     g_margin
     g_width
     n_defaultWidth
     t_snap
     l_namelist
     )
     => d_paramShapeId
```

Description

Defines a path that has its vertices determined by a parameter of the cell. When you place the Pcell, you enter a coordinate string that is used as the vertices of the parameterized path. You can use this function more than once in a cellview if all paths have the same vertices.

Arguments

d_pathId	Database ID of a path that has its vertices determined by a parameter of the cell.
S_param	Name of the parameter controlling the vertices of the path. By default, this parameter is `coords`.
g_margin	SKILL expression for the margin by which the path is offset from the coordinate list defined by parameter.
g_width	SKILL expression for the width of the path.
n_defaultWidth	Value to use for the width if *width* evaluates to a nonnumerical value.
t_snap	Snap used by the editor when digitizing path vertices to place an instance of this cell. Valid Values: `orthogonal`, `anyAngle`, `diagonal`, `L90`
l_namelist	List of symbols used in the *g_width* SKILL expression. These become parameters of the Pcell.

Value Returned

d_paramShapeId	Group ID used to record details of the parameterized paths.

Examples

```
pcDefineParamPath( path 'coords 2 'gateWidth 1-5 "orthogonal" list( 'gateWidth ) )
```

Defines `path` as a parameterized path that takes its vertices from the value of the parameter `coords` offset by `2`, with a width equal to the value of the `gateWidth` parameter or 1–5 if `gateWidth` evaluates to a nonnumerical value. It tells the editor to use `orthogonal` snap when entering vertices for the path and declares `gateWidth` as a parameter of the cell.

pcDefineParamPolygon

```
pcDefineParamPolygon(
     d_polygonId
     S_param
     g_margin
     t_snap
     l_namelist
     )
     => d_paramShapeId
```

Description

Defines a polygon that has its vertices determined by a parameter of the cell. When you place the Pcell, you enter a coordinate string that is used as the vertices of the parameterized polygon. You can use this function more than once in a cellview.

Arguments

d_polygonId	The database ID of the polygon that has its vertices determined by a parameter of the cell.
S_param	Name of the parameter controlling the vertices of the polygon. By default, this parameter is `coords`.
g_margin	SKILL expression for the amount to enlarge or shrink the vertices of the polygon as compared to the coordinate list in the parameter.
t_snap	Snap used by the editor when digitizing polygon vertices to place an instance of this cell. Valid Values: `orthogonal`, `anyAngle`, `diagonal`, `L90`
l_namelist	List of symbols used in the *g_margin* SKILL expression. These become parameters of the Pcell.

Value Returned

d_paramShapeId	Group ID used to record details of the parameterized polygons.

Examples

```
pcDefineParamPolygon( polygon 'coords '(-margin) "orthogonal" list( 'margin ) )
```

Defines *d_polygonId* as a parameterized polygon that takes its vertices from the value of the parameter coords, shrunk by the value of the *g_margin* parameter. It tells the editor to use orthogonal snap when entering vertices for the polygon and declares margin as a parameter of the cell.

pcDefineParamProp

```
pcDefineParamProp(
     d_cvId
     t_name
     g_expr
     )
     => t / nil
```

Description

Defines a parameterized property that can be accessed using a SKILL procedure. You can use this property to store any value within the Pcell.

Arguments

d_cvId	The database ID of the cellview in which the parameter applies.
t_name	Name of the parameterized property.
g_expr	SKILL expression for the property value.

Value Returned

t	Returned if the parameterized property is created.
nil	Returned if the parameterized property is not created.

Examples

```
pcDefineParamProp(cv "myProp" 'length)
```

Defines a property name `myProp` to be a parameterized property of the cellview `cv`. The property `myProp` takes its value from the parameter `length` and is evaluated when the instance is created.

pcDefineParamRect

```
pcDefineParamRect(
     d_rectangleId
     S_param
     g_margin
     l_namelist
     )
     => d_paramShapeId
```

Description

Defines a rectangle that has its vertices determined by a parameter of the cell. When you place the Pcell, you enter two coordinates that are used as the vertices of the parameterized rectangle. You can use this function more than once in a cellview.

Arguments

d_rectangleId	The database ID of the rectangle that has its vertices determined by a parameter of the cell.
S_param	Name of the parameter controlling the vertices of the rectangle. By default, this parameter is `coords`.
g_margin	SKILL expression for the amount to enlarge or shrink the vertices of the rectangle as compared to the coordinate list in the parameter.
l_namelist	List of symbols used in the *g_margin* SKILL expression. These become parameters of the Pcell.

Value Returned

d_paramShapeId	Group ID used to record details of the parameterized rectangles.

Examples

```
pcDefineParamRect(rect 'coords '(-margin) list('margin))
```

Defines `rect` as a parameterized rectangle that takes its vertices from the value of the parameter `coords`, reduced by the value of the `margin` parameter. It also declares `margin` as a parameter of the cell.

pcDefineParamRefPointObject

```
pcDefineParamRefPointObject(
     g_objects
     S_param
     l_refpoint
     )
     => d_refPointId
```

Description

Specifies that the location of an object or group of objects in the instance is determined by the location of a reference point that is a parameter of the cell. The objects in the instance have the same relationship to the reference point parameter as the objects in the master cellview have to the corresponding reference point in the master cellview.

Arguments

g_objects	List of objects whose locations are to be determined by the parameterized reference point.
S_param	Name of the parameter for the parameterized reference point. Valid Values: any string or symbol
l_refpoint	Coordinates of the reference point in the master cellview.

Value Returned

d_refPointId	Group ID used to record the details of the parameterized reference point.

Examples

```
pcDefineParamRefPointObject(list(pin) 'pinPoint centerBox (pin~>bBox))
```

Causes the object `pin` to be located relative to the coordinate parameter `pinPoint`. The reference point for `pinPoint` in the master cellview is the center point of the `pin` object.

pcDefineParamSlot

```
pcDefineParamSlot(
     g_type
     g_value
     g_pcIsParamSlot
     )
     => l_paramDefDPL
```

Description

Constructs a parameter DPL, which is used to define a parameter slot of the specified device.

Arguments

g_type	A string defining the parameter type. Valid values are `string`, `int`, `float`, and `ILList`.
g_value	The value of the parameter slot.
g_pcIsParamSlot	A Boolean, which indicates whether it is a parameter slot. Default value is `t`.

Value Returned

l_paramDefDPL	The created parameter DPL.

Examples

```
defclass( CORE ( pcParamClass )
    (
         (cyanW @initform pcDefineParamSlot("float" 0.6))
         (cyanL @initform pcDefineParamSlot("float" 0.2))
         (greenW @initform pcDefineParamSlot("float" 0.2))
         (greenL @initform pcDefineParamSlot("float" 0.8))
         (coreBBox @initarg coreBBox)
    )
)
```

Constructs a parameter DPL, which is used to define a parameter slot of `pcParamClass`.

pcDefinePathRefPointObject

```
pcDefinePathRefPointObject(
     l_objects
     S_param
     t_endpoint
     )
     => d_refPointId
```

Description

Specifies that the location of an object or group of objects in the instance be determined by the location of the endpoint of a parameterized path. The objects in the instance have the same relationship to the endpoint of the digitized path in the instance as the objects in the master cellview have to the corresponding endpoint of the parameterized path in the master cellview.

Arguments

l_objects	List of objects whose locations are to be determined by the endpoint of a parameterized path.
S_param	Name of the parameter controlling the parameterized path whose endpoint determines the location of the objects. By default, this parameter is `coords`.
t_endpoint	Endpoint of the path that determines the location of the objects. Valid Values: `first`, `last`

Value Returned

d_refPointId	Group ID used to store the details of the parameterized reference point.

Examples

```
pcDefinePathRefPointObject( list ( via ) 'coords "first" )
```

Causes the object `via` to be located relative to the first vertex of the `coords` parameter. The `coords` parameter controls the vertices of a parameterized path.

pcDefinePCell

```
pcDefinePCell(
     l_cellIdentifier
     l_formalArgs
     body_of_code
     )
     => d_cellViewId / nil
```

Description

Creates parameterized (super) master cellview. This function enables you to pass a SKILL definition for a Pcell including a list of its parameters.

Also, refer to dbDefineProc in the *Virtuoso Studio Design Environment SKILL Reference*.

Arguments

l_cellIdentifier	List containing the library database ID, cell name, view name, and view type. View type is optional and defaults to `maskLayout`. Valid Values: `maskLayout`, `schematic`, `schematicSymbol`
l_formalArgs	The parameter declaration section, containing a list of input parameters and their default values, all enclosed in parentheses. Each element of the list is itself a list containing a parameter symbol, its (optional) data type, and a default value. **Do not** use the `list` keyword, but enclose the parameters in parentheses instead. In addition, The *l_formalArgs* argument to `pcDefinePCell` can include an expression. However, it is the value of that expression and not the expression itself, which gets associated as a default value of the corresponding parameter. As a result, such expression cannot be retrieved later on once the Pcell has been compiled (i.e. super master created).

You can specify *l_formalArgs* as:

Format 1:
(S_paramName tS_paramType g_paramValue)

Format 2:
(S_paramName g_paramValue)

where,

- *S_paramName*: A symbol of the parameter name.
- *tS_paramType*: Either a string or a symbol to represent the parameter type of a Pcell parameter.
  You may or may not enclose the parameter type within quotes. Valid values include Boolean, int, float, and ilList. It is optional to include the parameter type (see the ***Caution*** note).
- *g_paramValue*: Default value of a defined parameter. The value specified must match the specified parameter type if the specified parameter type is defined as a Pcell parameter list.

Cadence recommends following the first format to define a Pcell parameter because this format can be comprehended clearly both by the graphical Pcell application and the end user of Pcell.

Examples

```
( Switch1 "float" 0.625 )
( Switch2 float 0.625 )
( Switch3 0.625 )
```

For Boolean arguments, you “must” specify the data type. It helps you avoid any issues with the accurate parameter type checking, which is enforced in OpenAccess. If you specify the default value of an argument as 'nil' without specifying the data type, the argument value might be interpreted as the integer zero instead of the boolean nil (false).

Consider specifying the data type of a Boolean argument as shown in the following example:

```
(is_orthogonal_boolean_arg "boolean" nil)
```

where, the data type of the `is_orthogonal_boolean_arg` argument has been specified as `Boolean` and the default value of this Boolean argument has been set to `nil`.

`body_of_code`

SKILL code for creating geometries and pins. You must provide code here. Enclose the code in a `let` or `prog` statement. If you use variables in the code, define them at the beginning of the `let` or `prog` statement. Using `let` gives faster performance than `prog`; `prog` allows multiple exits while `let` exits only at its end. Optionally, from within this code, you can call SKILL functions, application functions, and your own user-defined functions.

Calls to functions from within the *`body_of_code`* section are part of the Pcell, but the functions themselves are **not** part of the Pcell. These functions must be available each time the Pcell is evaluated. If a called function is not available when a Pcell is evaluated, the Pcell fails.

If you want the system to automatically load functions when it opens your library, include the functions in your `libInit.il` file. **Do not** load a file from within a Pcell.

Value Returned

`d_cellViewId`	The cellview ID of the parameterized (super) master is returned if the (super) master is created.
`nil`	Returned if the parameterized (super) master cellview is not created.

Examples

```
pcDefinePCell(
 list( ddGetObj("tutorial") "ask" "layout"
 ) ; end of list for first argument
 (
    (layer string "metal1")
    (width float 3.0)
    (length float 6.0)
  ) ; end of pcell parameters
  let(
        () ; no local variables in this example
        rodCreateRect (
```

```
                        ?name       "minMetal"
                        ?width      width
                        ?layer      layer
                        ?length     length
      ) ; end of rodCreateRect
    ) ; end of body_of_code let
 ) ; end of pcDefinePCell
```

This example creates a rectangle named `minMetal` on the `metal1` layer, with a width equal to 3 and a length equal to six.

Related Topics

Safety Rules for Creating SKILL Pcells

Compile to Pcell

pcHIDefineParamCell

pcDefinePCell

pcDefinePPCell

```
pcDefinePPCell(
     l_cellIdentifier
     l_overrideCodeParams
     l_defaultCodeParams
     ls_formalArgs
     body_of_code
     )
     => d_cellViewId / nil
```

Description

Creates a parameterized Pcell master cellview (P^2cell) that enables you to pass the SKILL definition for a Pcell, including a list of its arguments, similar to the pcDefinePCell function. This function also accepts a code argument declaration list and a code argument override list, where the code parameter can be used in the Pcell code body to create multiple Pcell super masters from a single Pcell code.

Within `pcDefinePPCell`, the override code parameter is always a simple variable, which receives the input provided by the caller of `pcDefinePPCell`. This will always be the use model of P^2cell because there is always a driver function that calls it. The function should not be used as a top-level function (not inside a procedure) because that will lose the purpose of the new parameterization feature.

Arguments

l_cellIdentifier	A list that contains the library database ID, cell name, view name, and view type. View type is optional and defaults to `maskLayout`. Valid view type values: `maskLayout`, `schematic`, `schematicSymbol`

l_overrideCodeParams A SKILL list in the DPL format. Each parameter within the DPL is represented by a pair of *s_codeParamName* and *g_codeParamValue*. Only the value can be a SKILL expression.

Syntax:

```
list( nil S_codeParamName
g_codeParamValue ... )
```

Example:

```
list( nil 'param1 "value1" 'param2 t
'param3 width*3 )
```

- *S_codeParamName*: A symbol or a string of the name of code parameter. It needs to match a declared code parameter name. A warning is issued for a mismatched name and the default value is used instead.
- *g_codeParamValue*: A SKILL expression that evaluates the given value to obtain the value of each code parameter. Within the `pcDefinePPCell` function, the override code argument is always a simple variable that receives the input provided by the caller of `pcDefinePPCell`.

  The caller of `pcDefinePPCell` can construct the value of a code parameter using any method as long as the format of the result specified by the override code parameter is in the DPL format. This helps you to assign and update the values. It is expected this will be modified later to generate multiple Pcell super masters from one P^2cell definition.

l_defaultCodeParams A list that contains one or many code parameter specification. Each code parameter specification is defined in the following format:

(*S_codeParamName* [*S_codeParamType*] *g_codeParamValue*)

- *S_codeParamName*: A symbol or a string of the name of code parameter. It needs to match a declared code parameter name. A warning is issued for a mismatched name and the default value is used instead.
- *S_codeParamType*: A string or a symbol defining the value type of the code parameter.

  Valid values: `Boolean`, `int`, `float`, `string`, and `ilList`. It is optional to include the code parameter type except for Boolean. For Boolean code parameter, you must specify the data type to avoid ambiguity against the `ilList` type.
- *g_codeParamValue*: A SKILL expression that evaluates the given value to obtain the value of each code parameter. Within the `pcDefinePPCell` function, the override code argument is always a simple variable that receives the input provided by the caller of `pcDefinePPCell`.

  The caller of `pcDefinePPCell` can construct the value of a code parameter using any method as long as the format of the result specified by the override code parameter is in the DPL format. This helps you to assign and update the values. It is expected this will be modified later to generate multiple Pcell super masters from one P^2cell definition.

The declared code parameter is in a simple list format. This format is consistent with the declaration of Pcell parameters used by pcDefinePCell.

This declaration cannot contain another SKILL expression inside it.

Note: Like the override code parameter, the entire declaration itself can be defined outside `pcDefinePPCell`, but it hides the declaration from the Pcell code and is therefore not recommended.

`ls_formalArgs`

A list of parameters defining a Pcell parameter set. There are the following two ways to define the Pcell parameter set.

- Define a Pcell parameter through an explicit list of parameters, same style as the pcDefinePCell function.
- Call a user-defined function, which returns a Pcell parameter set. When P^{2}cell parser detects the Pcell parameters by calling a user-defined function, Parser executes the user function and pass the newly created super master ID as its argument.

`body_of_code`

SKILL code for creating geometries and pins. You must provide code here. Enclose the code in a `let` or `prog` statement. If you use variables in the code, define them at the beginning of the `let` or `prog` statement. Using `let` gives faster performance than `prog`; `prog` allows multiple exits while `let` exits only at its end. Optionally, from within this code, you can call SKILL functions, application functions, and your own user-defined functions.

Calls to functions from within the *`body_of_code`* section are part of the Pcell, but the functions themselves are **not** part of the Pcell. These functions must be available each time the Pcell is evaluated. If a called function is not available when a Pcell is evaluated, the Pcell fails.

If you want the system to automatically load functions when it opens your library, include the functions in your `libInit.il` file. **Do not** load a file from within a Pcell.

You can access the name and value of any code parameter using `pcGetCodeParamNames` and `pcGetCodeParamValue` functions.

Value Returned

d_cellViewId	The cellview ID of the parameterized super master if the Pcell is successfully compiled.
nil	If the parameterized super master cellview is not created.

Examples

```
procedure( createPmos(xVal)
    let((libName cellName ppcDef overrideCodeParams )
        ; Setup library and cell name, and the override code parameter
        ; values
        libName = "PcellDemo"
        cellName = "pmos"
        overrideCodeParams = list( nil
            'layerName "Via2" 'rectWidth 3*xVal 'createGate t)

        ; Calls myMakeMos to create one Pcell super master
        myMakeMos( libName cellName "layout" "maskLayout"
            overrideCodeParams)


        ; Update the cell name and one of the override code parameter value
        cellName = "pmos_ng"
        overrideCodeParams->createGate = nil


        ; Calls myMakeMos to create another Pcell super master
        myMakeMos( libName cellName "layout" "maskLayout"
            overrideCodeParams)
    )
)


procedure( myMakeMos(libName cellName viewName viewType @optional
        (overrideCPs nil))
    let((smId)
        ; Use pcDefinePPCell to create a Pcell that accepts code parameters
        smId = pcDefinePPCell(
            ; Define super master's identification
            list(ddGetObj(libName) cellName viewName viewType )


            ; Define variable to accept override code parameters
            overrideCPs

            ; Define code parameters
            (   ( layerName string "Metal")
                ( rectWidth float 1.0  )
                ( createGate boolean nil )
            )


            ; Define Pcell parameters
            (   (dd1 "none" )
                (sd1 "none" )
                (w      5.0)
                (l      1.0)
```

```
                (fingers 1 )
            )

            ; Define Pcell code
            let( (cv rect rectW createGate)
                cv = pcCellView
                sd1 = cv~>parameters~>sd1
                dd1 = cv~>parameters~>dd1
                w = aelNumber(cv~>parameters~>w)
                l = aelNumber(cv~>parameters~>l)
                fingers = cv~>parameters~>fingers

                ; Leverage code parameter to create different
                ; contents for a Pcell super master
                createGate = pcGetCodeParamValue(cv "createGate")
                when( createGate
                    layerName = pcGetCodeParamValue(cv "layerName")
                    rectW = pcGetCodeParamValue(cv "rectWidth")
                    rect = dbCreateRect(cv
                        list(layerName "drawing")
                        list( 2:2 3:2+round(rectW))))
            )
        )
        smId
    )
)
```

This example develops multiple Pcell super masters by using the P^2cell method.

Related Topics

Safety Rules for Creating SKILL Pcells

pcGetCodeParamNames

pcGetCodeParamValue

pcDefineRepeat

```
pcDefineRepeat(
     d_cvId
     l_shapes
     l_namelist
     g_stepX
     g_stepY
     g_repeatX
     g_repeatY
     g_stretchX
     g_stretchY
     g_adjustX
     g_adjustY
     t_direction
     )
     => d_repeatId
```

Description

Defines a repetition parameter to be applied to specified objects. Objects can be repeated in the X direction, Y direction, or both. If the value for the repetition direction, *t_direction*, is `horizontal` and `vertical`, the `pcDefineRepeat` function creates a two-dimensional array.

Arguments

d_cvId	The database ID of the cellview in which the parameter applies.
l_shapes	List of the shapes replicated.
l_namelist	List of the symbols referenced in the step or repeat expressions. These become parameters of the Pcell.
g_stepX	SKILL expression specifying stepping distance in the horizontal direction.
g_stepY	SKILL expression specifying stepping distance in the vertical direction.
g_repeatX	SKILL expression specifying the number of times objects are repeated in the horizontal direction.
g_repeatY	SKILL expression specifying the number of times objects are repeated in the vertical direction.
g_stretchX	Name of the vertical, dependent stretch control line associated with this repetition.

g_stretchY	Name of the horizontal, dependent stretch control line associated with this repetition.
g_adjustX	SKILL expression specifying the adjustment from the reference dimension to be applied to the vertical, dependent stretch control line.
g_adjustY	SKILL expression specifying adjustment from the reference dimension to be applied to the horizontal, dependent stretch control line.
t_direction	Direction of repetition. Valid Values: `horizontal`, `vertical`, `horizontal and vertical`

Value Returned

d_repeatId	Group ID used to store details of the repetition.

Examples

```
pcDefineRepeat(
    cv  ; cv is defines as cellview ID
    figs ; list of figures to repeat
    list('numGates 'cellPitch)
    'cellPitch
    nil
    '(numGates-1)
    nil
    'gateStretch
    nil
    '(pcFix(pcRepeat - 1) * pcStep)
    nil
    'horizontal
) ; close for pcDefineRepeat
```

`numGates` and `cellPitch` are parameters of the Pcell

Defines a horizontal repetition in cellview `cv` affecting the objects listed in `figs`. The horizontal stepping distance is `cellPitch`. The number of horizontal repetitions is `numGates-1`.

This example declares `numGates` and `cellPitch` as parameters of the Pcell. It also names `gateStretch` as a dependent stretch control line for the repetition, with an adjustment from the reference dimension defined by the expression `pcFix(pcRepeat - 1) * pcStep`.

pcDefineSteppedObject

```
pcDefineSteppedObject(
     g_objects
     S_param
     g_step
     g_startOffset
     g_endOffset
     l_namelist
     )
     => d_stepObjectId
```

Description

Defines an object or group of objects to be repeated along the length or perimeter of a parameterized shape that has already been defined in the Pcell. The selection filter prevents you from selecting stretch control lines, parameterized shapes, or objects in other repeat-along-shape groups.

Arguments

g_objects	List of objects repeated along the length or perimeter of a parameterized shape.
S_param	Name of the parameter controlling the parameterized shape along which objects are repeated. By default, this parameter is `coords`.
g_step	SKILL expression for the stepping distance between repetitions.
g_startOffset	SKILL expression for the space left between the first vertex of the parameterized shape and the first repeated object.
g_endOffset	SKILL expression for the space left between the last vertex of the parameterized shape and the last repeated object.
l_namelist	List of symbols used in *g_step*, *g_startOffset*, or *g_endOffset* expressions. These become parameters of the Pcell.

Value Returned

d_stepObjectId	Group ID used to record the details of the repetition along the parameterized shape.

Examples

```
pcDefineSteppedObject(list(via1 via2) 'coords 'viaPitch 1.25 1.25 list('viaPitch))
```

Specifies that objects `via1` and `via2` are to be repeated along the parameterized shape controlled by `coords`, using a stepping distance determined by the parameter `viaPitch`, with a gap of `1.25` between the start of the vertices and the first repetition and between the last repetition and final vertex of parameterized shape. It also declares `viaPitch` as a parameter of the Pcell.

pcDefineStretchLine

```
pcDefineStretchLine(
     d_lineId
     g_paramExpr
     t_direction
     f_defval
     f_minval
     f_maxval
     g_stretchRepeated
     )
     => d_StretchId
```

Description

Defines a stretch control line used to control stretching in the X direction or Y direction. Objects repeated in the direction parallel to the stretch direction can be set to stretch.

Arguments

d_lineId	The database ID of the line used as the stretch control line. By default, this line is drawn on the layer `stretch`.
g_paramExpr	SKILL expression or symbol controlling the stretch.
t_direction	Direction of the stretch. Valid Values: `right`, `left`, `rightAndLeft`, `up`, `down`, `upAndDown`
f_defval	Default value (reference dimension) for the stretch.
f_minval	Minimum value for the stretch.
f_maxval	Maximum value for the stretch. Use `nil` if no maximum is specified.
g_stretchRepeated	Boolean expression indicating whether to stretch shapes that are repeated in the direction parallel to the stretch.

Value Returned

d_StretchId	Group ID used to store the details of the stretch parameter.

Examples

```
pcDefineStretchLine(stretchLine 'nfetWidth, "right" 1.25 1.25 25 nil)
```

Defines `stretchLine` as a stretch control line controlled by the parameter `nfetWidth`. The stretch direction is `right`. The default and the minimum values are `1.25`, and the maximum value is `25`. Horizontally repeated shapes are not stretched.

pcDeleteCondition

```
pcDeleteCondition(
     d_groupId
     )
     => t / nil
```

Description

Deletes a previously defined conditional inclusion parameter.

Arguments

d_groupId	Group ID of the conditionally included objects.

Value Returned

t	Returned if the parameter is deleted.
nil	Returned if *d_groupId* is not a conditional inclusion parameter.

Examples

```
foreach(cond pcGetCondition(cv) pcDeleteCondition(cond))
```

Deletes all conditional inclusion parameters in the cellview cv.

pcDeleteParam

```
pcDeleteParam(
     d_cvId
     t_paramName
     )
     => t / nil
```

Description

Deletes the definition of the parameter identified by *t_paramName*. It is recommended to use pcHIEditParameters instead of this function.

You must have defined the named parameter.

Arguments

d_cvId	The database ID of the cellview in which the parameterized property should be deleted.
t_paramName	Name of the parameter in quotes.

Value Returned

t	Returned if the parameter definition is deleted.
nil	Returned if *t_paramName* is not a valid parameter name.

Examples

```
pcDeleteParam( cellviewId "width" )
```

Deletes the parameter named width.

pcDeleteParamLayer

```
pcDeleteParamLayer(
     d_groupId
     )
     => t / nil
```

Description

Deletes a parameter associating a set of shapes with a layer parameter.

You must have defined the layer parameter.

Arguments

d_groupId	Group ID of the shapes assigned the layer parameter you want to delete.

Value Returned

t	Returned if the parameter is deleted.
nil	Returned if *d_groupId* is not a valid identifier for a layer parameter.

Examples

```
pcDeleteParamLayer(diffLayerParam)
```

Deletes the layer parameter identified by `diffLayerParam`.

pcDeleteParamProp

```
pcDeleteParamProp(
     d_cvId
     t_propname
     )
     => t / nil
```

Description

Deletes a parameterized property in the cellview *d_cvId*.

Arguments

d_cvId	The database ID of the cellview in which the parameterized property should be deleted.
t_propname	Name of the parameterized property.

Value Returned

t	Returned if the parameterized property is deleted.
nil	Returned if no parameterized property is defined with the given name or the parameterized property cannot be deleted.

Examples

```
pcDeleteParamProp( cv "myProp" )
```

Deletes the parameterized property named myProp.

pcDeleteParamShape

```
pcDeleteParamShape(
     d_memberId
     )
     => t / nil
```

Description

Deletes a parameterized shape directive, causing the specified shape to revert back to a regular (nonparameterized) shape.

You must have defined the shape as a parameterized path, polygon, or rectangle.

Arguments

d_memberId	The database ID of the shape whose parameter you want to delete.

Value Returned

t	Returned if the parameter is deleted.
nil	Returned if the shape is not defined as a parameterized path, polygon, or rectangle.

Examples

```
foreach(shape allMyParamShapes pcDeleteParamShape(shape))
```

Deletes all shape parameters in the list `allMyParamShapes`.

pcDeleteRefPoint

```
pcDeleteRefPoint(
     d_groupId
     )
     => t / nil
```

Description

Deletes a reference point parameter. You can use this command to delete either a reference point defined relative to a parameter of the cell or a reference point defined relative to a parameterized path endpoint.

Note: You must have defined the reference point parameters.

Arguments

d_groupId	The database ID of the parameterized reference point you want to delete.

Value Returned

t	Returned if the reference point parameter is deleted.
nil	Returned if *d_groupId* is not a valid identifier for a parameterized reference point.

Examples

```
pcDeleteRefPoint( gateContactRefPt )
```

Deletes the parameterized reference point defined by the symbol `gateContactRefPt`.

pcDeleteRepeat

```
pcDeleteRepeat(
     d_groupId
     )
     => t / nil
```

Description

Deletes the repetition parameter *d_groupId*.

Arguments

d_groupId	The database ID of the repetition parameter you want to delete.

Value Returned

t	Returned if the parameter is deleted.
nil	Returned if *d_groupId* is not a repeat identifier.

Examples

```
foreach( repeat pcGetRepeats ( cv ) pcDeleteRepeat ( repeat ) )
```

Deletes all repetition parameters in the cellview cv.

pcDeleteSteppedObject

```
pcDeleteSteppedObject(
     d_groupId
     )
     => t / nil
```

Description

Deletes the repetition-along-shape parameter *d_groupId*.

Arguments

d_groupId	The database ID of the repetition along a shape as returned by `pcDefineSteppedObject`.

Value Returned

`t`	Returned if the parameter is deleted.
`nil`	Returned if *d_groupId* is not a repetition-along-shape identifier.

Examples

```
foreach( obj allMySteppedObjs pcDeleteSteppedObject ( obj ) )
```

Deletes all repetition-along-shape parameters in the list `allMySteppedObjs`.

pcDraw

```
pcDraw(
     g_device
     )
     => t / nil
```

Description

Creates database objects for the specified SKILL++ Pcell class. This method is implemented for each SKILL++ Pcell class. `pcDraw` is called by the Pcell source code in `pcDefinePCell` for the associated Pcell.

Arguments

g_device	A SKILL++ Pcell class object that is inherited from class `pcParamClass`.

Value Returned

`t`	If the database objects are created for a specified SKILL++ Pcell class.
`nil`	If the database objects are not created.

Examples

```
defmethod( pcDraw ((device RING))
    let((cv ringS ringW coreBBox llx lly urx ury pts ring)
        ringS = pcGetParamSlotValue(device 'ringS)
        ringW = pcGetParamSlotValue(device 'ringW)
            coreBBox = getCoreBBox(device)
            llx = xCoord( lowerLeft(coreBBox))
            lly = yCoord( lowerLeft(coreBBox))
            urx = xCoord( upperRight(coreBBox))
            ury = yCoord( upperRight(coreBBox))
                pts = list(
                    llx-ringS:lly-ringS             ; points on inner edges
                    urx+ringS:lly-ringS
                    urx+ringS:ury+ringS
                    llx-ringS:ury+ringS
                    llx-ringS:lly-ringS-ringW   ; extending to outer edge
                    llx-ringS-ringW:lly-ringS-ringW ; points on outer edges
                    llx-ringS-ringW:ury+ringS+ringW
                    urx+ringS+ringW:ury+ringS+ringW
                    urx+ringS+ringW:lly-ringS-ringW
                    llx-ringS:lly-ringS-ringW)
                    ; layer is chosen for its color's visibility
```

```
                    cv   = slotValue(device 'cvId)
                    ring = dbCreatePolygon( cv list("Via2" "drawing") pts)
                callNextMethod()
        )
)
```

Creates the database objects from SKILL++ Pcell class object, `RING`.

pcExprToProp

```
pcExprToProp(
     txfl_argument
     )
     => l_typeValue / nil
```

Description

Evaluates the argument and returns a list containing the data type and value of the result.

Arguments

txfl_argument	String, integer, floating-point number, list, or combination of data types forming a valid expression, that specify the argument. Valid Values: String, integer, floating-point number, list, or a combination of data types that form a valid expression.

Value Returned

l_typeValue	List containing the following two elements: data type and value, where the data type can be a string, integer, or floating-point number.
nil	Returned if *txfl_argument* has an invalid data type or contains an invalid expression.

Examples

```
a = 1.5
b = 1.6
pcExprToProp(a + b + 3)=> ("float" 6.1)
```

Returns the data type (floating-point number) and value (6.1) of the result of evaluating the specified argument.

```
pcExprToProp( '( 2 3 ) ) => "ilList" (2 3)
```

Returns the data type (list) and value (a list containing two numbers, 2 and 3).

pcFilterPoints

```
pcFilterPoints(
     l_pointList
     )
     => l_pointList / nil
```

Description

Determines whether the specified list of coordinates represents a manhattan shape.

Arguments

l_pointList	List of coordinates in one of the following formats:
	`list( '( x1 y1 ) '( x2 y2 ) ... '( xn yn ) )`
	`'( x1:y1 x2:y2 ... xn:yn )`

Value Returned

`t`	Returns the list of coordinates when it represents a manhattan shape.
`nil`	Returned if the list of coordinates does not represent a manhattan shape or the function did not complete successfully.

Examples

```
pcFilterPoints( list( '( 0 0 ) '( 0 5 ) '( 5 5 ) '( 5 1 ) '( 7 1 ) '( 7 0 ) )
    => ((0 0) (0 5) (5 5) (5 1) (7 1) (7 0))
```

The list of coordinates does represent a manhattan shape.

pcFix

```
pcFix(
     n_num
     [ f_precision ]
     )
     => x_result
```

Description

Converts a number to an integer in the format *fixnum*. When *n_num* is *close* to a whole number, the system keeps the integer part of the number and adds a single decimal place equal to zero. Here, *close* means the value of the number is within the range of plus or minus the value of *f_precision* of the integer part of the number specified by *n_num*. When the value is not within this range, the function allows the system to use the value in the first decimal place to round the *n_num* to an integer in the format *number*.0; the system ignores all other decimal places. This function is useful for correcting the round-off approximation that can occur with floating-point numbers that are stored in 32 or 64 bits.

Arguments

n_num	Any number you want to convert.
f_precision	Floating-point number specifying number of decimal places to the right of the decimal point to use as a range for determining whether *n_num* is close to a whole number. Default: 0.001

Value Returned

x_result	A number with one decimal place that is equal to zero. If *n_num* does not contain a number, an error occurs; look in the CIW or at the CDS.log file for a message about the error.

Examples

If you want to test for the condition *x equals 6.0*, you need to use pcFix to correct a possible rounding problem that could occur with a floating-point number. Otherwise, your condition might never be satisfied, because x might never be equal to 6.0; instead, it might be a close value, such as 5.999999.

The following code shows how to use `pcFix` to correct floating-point rounding errors:

```
if( pcFix( x ) == 6.0
    ; perform conditional operations here
) ;endif
```

If you used the code shown below instead, your condition might never be satisfied:

```
x = 6.0
if ( x == 6.0
    ; perform conditional operations here
) ;endif
```

Additional Information

If the condition you want to test is `x = 15.0`, use the `pcFix` function to ensure that results in the range from 14.999 to 15.999 are converted to 15.0. This implies the following:

When *x* equals	`pcFix( x )` returns
14.998	14.0
14.999	15.0
15.0	15.0
15	15.0
15.998	15.0
15.999	16.0

The following shows the results of specifying the optional *f_precision* argument:

When *x* equals	And `f_precision` equals	Then `pcFix` returns
14.998	0.001	14
14.998	0.005	15
14	0.001	14
14	0.005	14
14.0	0.001	14
14.0	0.005	14
14.999	0.0001	14

14.999	0.001	15

Related Topics

pcRound

pcExprToString

pcTechFile

Recommended, Supported SKILL Functions for Pcells

pcGetBendAngle

```
pcGetBendAngle(
     l_pointList
     )
     => x_signedDegrees / nil
```

Description

Determines whether the specified list of three coordinates represents a bend to the left (+90 degrees) or to the right (-90 degrees). The three coordinates must define a 90 degree angle.

Arguments

l_pointList	List of three coordinates in one of the following formats:
	`list( '( x1 y1 ) '( x2 y2 ) '( x3 y3 ) )`
	`'( x1:y1 x2:y2 x3:y3 )`

Value Returned

x_signedDegrees	Returns the positive number 90 when the three coordinates represent a bend to the left; returns the number -90 when the three coordinates represent a bend to the right.
`nil`	Returned if the three coordinates do not represent a bend to the right or to the left, or if the function did not complete successfully.

Examples

```
pcGetBendAngle( 0:1 1:0 1:1 ) => 90
```

The three coordinates represent a bend to the left.

pcGetCodeParamNames

```
pcGetCodeParamNames(
     d_cellViewId
     )
     => l_codeParamNames / nil
```

Description

Returns a list of all the defined code parameter names for the specified design ID.

Arguments

d_cellViewId	A Pcell super master ID or Pcell sub master ID. If you specify a sub master ID, the function will obtain the defined code parameter names from its super master.

Value Returned

l_codeParamNames	A list of all the defined code parameter names of the specified cellview ID.
nil	No code parameter is defined for the specified cellview ID.

Examples

```
pcGetCodeParamNames( cvId )
=> ( "param_1" "param_2" "param_3" )
```

Returns all the defined code parameter names for `cvId`.

pcGetCodeParamValue

```
pcGetCodeParamValue(
     d_cellViewId
     S_codeParamName
     )
     => g_value / nil
```

Description

Returns the value of a specified code parameter.

Arguments

d_cellViewId	A Pcell super master ID or sub master ID. If cellView ID is a Pcell sub master ID, the function will obtain the code parameter from its super master.
S_codeParamName	The name of any defined code parameter.Type: Symbol or String.

Value Returned

g_value	Returns the value of the specified code parameter. The returned value can be integer, float, string, Boolean, or a list.
nil	Specified code not found.

Examples

```
createGateValue = pcGetCodeParamValue( cvId "createGate" )
=> t
```

In this example, `createGate` is a defined Boolean code parameter. It returns the value of the `createGate` code parameter.

pcGetConditions

```
pcGetConditions(
     d_cvId
     )
     => l_condlist / nil
```

Description

Returns a list of identifiers for conditional inclusion parameters in the specified cellview.

You must have defined the conditional inclusion parameters.

Arguments

d_cvId	The database ID of the cellview containing conditional inclusion parameters.

Value Returned

l_condlist	List of the object IDs used to store the details of the conditional inclusion parameters.
nil	Returned if *d_cvId* does not identify a cellview or if no conditional inclusion parameters are defined in the cellview.

Examples

```
condlist = pcGetConditions(cv)
```

Lists all conditional inclusion parameters defined in `cv`.

pcGetDefaultParamsFromClass

```
pcGetDefaultParamsFromClass(
     s_className
     )
     => l_paramList
```

Description

Displays the defined Pcell parameter list from the specified class name.

Arguments

s_className	A class symbol declared for a class name of a Pcell.

Value Returned

l_paramList	The parameter list that contains (*paramName paramType value*).

Examples

```
pcGetDefaultParamsFromClass('CORE)
```

Returns a defined Pcell parameter list from the specified class name, 'CORE'.

pcGetInheritParamDefn

```
pcGetInheritParamDefn(
     d_instId
     )
     => l_inherit / nil
```

Description

Returns an identifier for an inherited parameter in the specified cellview.

You must have defined the inherited parameter.

Arguments

d_instId	The database ID of the instance of the Pcell whose inherited parameter is to be retrieved.

Value Returned

l_inherit	Group ID used to record details of the inherited parameter.
nil	Returned if *d_instId* does not identify an instance as a Pcell or if there is no inherited parameter defined in it.

Examples

```
inherit = pcGetInheritParamDefn(inst)
```

Retrieves the inherited parameter defined in *d_instId*.

pcGetInheritParams

```
pcGetInheritParams(
     d_cvId
     )
     => l_inheritlist / nil
```

Description

Returns a list of identifiers for inherited parameters in the specified cellview.

You must have defined the inherited parameter.

Arguments

d_cvId	The database ID of the cellview whose inherited parameters you want listed.

Value Returned

l_inheritlist	List of object IDs used to record details of the inherited parameters.
nil	Returned if *d_cvId* does not identify a cellview or if there are no inherited parameters defined in the cellview.

Examples

```
inheritlist = pcGetInheritParams( cv )
```

Gets a list of all the inherited parameters defined in `cv`.

pcGetOffsetPath

```
pcGetOffsetPath(
     d_cvId
     l_vertex_points
     n_offset
     )
     => l_vertex_points / nil
```

Description

Applies the offset to each vertex of the path to create a list of vertices for a longer or shorter version of the path. Returns a list of points for the vertices of the offset version of the path. This function does not change the original path and does not create the offset version of the path

Arguments

d_cvId	The database ID of the cellview containing the path.
l_vertex_points	A list of lists identifying a path in the cellview, where each sublist specifies the coordinates for one vertex. Use the following format: `list( '( x y ) '( x y ) ... '( x y ) )`
n_offset	An integer or floating-point number.

Value Returned

l_vertex_points	A list of lists specifying the vertices of the offset version of the path, in the following format: `((x y) (x y) ... (x y))`
`nil`	The function did not execute successfully.

Examples

```
pcGetOffsetPath( cv list( '(6.0 6.0) '(10.0 6.0) '(10.0 16.0) '(20.0 16.0) ) 2.0 )
```

Creates and returns the following list of points, which are offset from the specified path by 2.0:

```
((6.0 8.0)
  (8.0 8.0)
  (8.0 18.0)
```

```
  (20.0 18.0)
)
```

pcGetOffsetPolygon

```
pcGetOffsetPolygon(
     d_cvId
     l_vertex_points
     n_offset
     )
     => l_vertex_points / nil
```

Description

Applies the offset to each edge of the polygon to create a list of the vertices for an oversized or undersized version of the polygon. Returns a list of points for the vertices of the offset version of the polygon. This function does not change the original polygon and does not create the offset version of the polygon

Arguments

d_cvId	The database ID of the cellview containing the polygon.
l_vertex_points	A list of lists identifying a polygon in the cellview, where each sublist specifies the coordinates for one vertex. Use the following format: `list( '( x y ) '( x y ) ... '( x y ) )`
n_offset	An integer or floating-point number; a positive number specifies an oversized polygon, a negative number specifies an undersized polygon.

Value Returned

l_vertex_points	A list of lists specifying the vertices of an oversized or undersized version of the polygon after the offset has been applied. The list is in the following format: `((x y) (x y) ... (x y))`
`nil`	The polygon was not found or the offset was not applied successfully.

Examples

```
pcGetOffsetPolygon( cv list( '(2.7 2.4) '(8.0 2.4) '(8.0 7.1) '(6.4 7.1)
'(6.4 4.3) '(2.7 4.3) ) 3.0 )
```

Oversize the polygon specified in the list by adding 3.0 to every edge, and returns the following points for the un-created oversized polygon:

```
((-0.3 -0.6)
   (11.0 -0.6)
   (11.0 10.1
   (3.4 10.1)
   (3.4 7.3)
   (-0.3 7.3)
)
```

pcGetParameters

```
pcGetParameters(
     d_cvId
     )
     => l_paramlist / nil
```

Description

Returns a list of parameters and their default values defined in the specified cellview.

You must have defined the parameters in this cellview.

Arguments

d_cvId	The database ID of the cellview containing the parameters.

Value Returned

l_paramlist	List of the parameter names and their default values. The list is in the form (*t_paramName g_defaultValue*).
nil	Returned if d_cvId does not identify a cellview or if there are no parameters defined in the cellview.

Examples

```
paramlist = pcGetParameters( cv )
```

Gets list of all parameters defined in cv.

pcGetParamLabelDefn

```
pcGetParamLabelDefn(
     d_labelId
     )
     => d_paramlabelId / nil
```

Description

Returns the parameterized label identifier resulting from a call to `pcDefineParamLabel` on the specified label.

You must have defined a parameterized label.

Arguments

d_labelId	The database ID of the label whose parameterized label is to be retrieved.

Value Returned

d_paramlabelId	Object ID of the parameterized label.
nil	Returned if *d_labelId* does not identify a label or there are no parameterized labels defined in it.

Examples

```
paramLabel = pcGetParamLabelDefn( label )
```

Retrieves parameterized labels defined in `label`.

pcGetParamLabels

```
pcGetParamLabels(
     d_cvId
     )
     => l_labellist / nil
```

Description

Returns a list of parameterized labels in the specified cellview.

You must have defined a parameterized label.

Arguments

d_cvId	The database ID of the cellview containing parameterized labels.

Value Returned

l_labellist	List of object IDs of the parameterized labels.
nil	Returned if *d_cvId* does not identify a cellview or if there are no parameterized labels defined in the cellview.

Examples

```
labellist = pcGetParamLabels( cv )
```

Gets a list of all the parameterized labels defined in `cv`.

pcGetParamLayerDefn

```
pcGetParamLayerDefn(
     d_instId
     )
     => l_layerId / nil
```

Description

Returns the identifier for a layer parameter of a specified shape.

You must have defined a layer parameter for this shape.

Arguments

d_instId	The database ID of the shape whose layer parameter is to be retrieved.

Value Returned

l_layerId	Object ID used to record details of a layer parameter.
nil	Returned if *d_instId* does not identify a shape or there are no layer parameters defined in it.

Examples

```
layer = pcGetParamLayerDefn( shape )
```

Retrieves the layer parameters defined in `shape`.

pcGetParamLayers

```
pcGetParamLayers(
     d_cvId
     )
     => l_layerlist / nil
```

Description

Returns a list of identifiers for layer parameters in the specified cellview.

You must have defined layer parameters for this cellview.

Arguments

d_cvId	The database ID of the cellview whose layer parameters you want listed.

Value Returned

l_layerlist	List of object IDs used to record details of layer parameters.
nil	Returned if *d_cvId* does not identify a cellview or if there are no layer parameters defined in the cellview.

Examples

```
layerlist = pcGetParamLayers( cv )
```

Gets a list of all the layer parameters defined in `cv`.

pcGetParamProps

```
pcGetParamProps(
     d_cvId
     )
     => l_proplist / nil
```

Description

Lists parameterized properties in the specified cellview.

You must have defined the parameterized properties.

Arguments

d_cvId	The database ID of the cellview containing parameterized properties.

Value Returned

l_proplist	List of object IDs of parameterized properties.
nil	Returned if *d_cvId* does not identify a cellview or if there are no parameterized properties defined in the cellview.

Examples

```
proplist = pcGetParamProps( cv )
```

Gets a list of all parameterized properties defined in `cv`.

pcGetParamShapeDefn

```
pcGetParamShapeDefn(
     d_instId
     )
     => d_paramShapeId / nil
```

Description

Returns an identifier for a parameterized shape parameter resulting from a call to `pcDefineParamPath`, `pcDefineParamPolygon`, or `pcDefineParamRect`.

You must have defined the parameterized shape.

Arguments

d_instId	The database ID of the shape whose parameterized shape parameter is to be retrieved.

Value Returned

d_paramShapeId	Group ID used to record details of the parameter.
nil	Returned if *d_instId* does not identify a shape or there are no parameterized shape parameters defined in it.

Examples

```
paramshape = pcGetParamShapeDefn( shape )
```

Retrieves parameterized shape parameters defined in `shape`.

pcGetParamShapes

```
pcGetParamShapes(
     d_cvId
     )
     => l_paramshapelist / nil
```

Description

Returns a list of identifiers for the parameterized shape parameters in the specified cellview.

You must have defined a parameterized shape for this cellview.

Arguments

d_cvId	The database ID of the cellview containing the shape parameters you want listed.

Value Returned

l_paramshapelist	List of the object IDs of the shape parameters.
nil	Returned if *d_cvId* does not identify a cellview or if there are no parameterized shapes defined in the cellview.

Examples

```
paramshapelist = pcGetParamShapes( cv )
```

Lists all shape parameters defined in `cv`.

pcGetParamSlotType

```
pcGetParamSlotType(
     g_device
     s_propName
     )
     => t_paramType
```

Description

Checks the type contained in the parameter slot `propName` of the given device. Parameter type is a string whose value is `int`, `float`, `string`, or `iLList`.

Arguments

g_device	A SKILL++ Pcell class object that is inherited from class `pcParamClass`.
s_propName	A parameter slot name of the given device.

Value Returned

t_paramType	The type contained in the parameter slot `propName` of the given device.

Examples

```
pcell =  makeInstance( 'CORE )
pcGetParamSlotType(pcell 'cyanW)
```

Returns the type of `cyanW` as `float`.

pcGetParamSlotValue

```
pcGetParamSlotValue
     g_device
     s_propName
     )
     => g_value
```

Description

Returns the value of the parameter slot, `propName`, of the specified device.

Arguments

g_device	A SKILL++ Pcell class object that is inherited from class `pcParamClass`.
s_propName	A Pcell slot name of the given device.

Value Returned

g_value	The value of the parameter slot, `propName`, of the given device.

Examples

```
pcell =  makeInstance( 'CORE )
pcGetParamSlotValue( pcell 'cyanW )
```

Returns the `cyanW` parameter slot value of the pcell.

pcGetPathRefPoint

```
pcGetPathRefPoint(
     d_cvId
     l_objectVertices
     l_pathVertices
     t_endpoint
     l_offset
     )
     => l_newObjVertices / nil
```

Description

Creates a list of coordinates representing the object specified by the *l_objectVertices* argument when offset from the specified endpoint of the parameterized path, where the offset is defined by *l_offset* and the path is defined *l_pathVertices*.

Arguments

d_cvId	The database ID of the cellview containing the parameterized path.
l_objectVertices	List of coordinates determining the object that is offset from the parameterized path, in one of the following formats: `list( '( x1 y1 ) '( x2 y2 ) ... '( xn yn ) )` `list( x1:y1 x2:y2 ... xn:yn )`
l_pathVertices	List of coordinates identifying the parameterized path, in one of the following formats: `list( '( x1 y1 ) '( x2 y2 ) ... '( xn yn ) )` `list( x1:y1 x2:y2 ... xn:yn )`
t_endpoint	String specifying an endpoint of the parameterized path. Valid Values: `"first"` or `"last"` Default: none
l_offset	List specifying the distance by which the object is offset from the endpoint of the parameterized path, where the elements of the list are integers or floating-point numbers. Valid Values: list of integers and/or floating-point numbers, or `nil` Default: none

Value Returned

l_newObjVertices	List of coordinates specifying the object as located in relationship to the specified endpoint of the parameterized path, offset by the specified distance. The list is in the following format: `( (x y) (x y) ... (x y) )` where x and y can be an integer or floating-point number.
`nil`	The function did not complete successfully.

Examples

```
obj =  list( 4.0:1.0 5.0:1.0 5.0:2.5 6.5:2.5 6.5:-0.5 4.0:-0.5 )
path = list( 0:0 0:9 )
ref =  pcGetPathRefPoint( geGetEditCellView() obj path "last" nil )
    => (        (4.0 10.0)
                (5.0 10.0)
                (5.0 11.5)
                (6.5 11.5)
                (6.5 8.5)
                (4.0 8.5)
          )
```

The example specifies the original object with the variable `obj` and the parameterized path with the variable `path`. The `pcGetPathRefPoint` function then determines the new coordinates for the object in relationship to the last endpoint of the path, with no offset.

pcGetRefPointDefn

```
pcGetRefPointDefn(
     d_objectId
     )
     => l_refPointId / nil
```

Description

Returns an identifier for the reference point parameter defined in the specified cellview.

You must have defined a reference point parameter for the cellview.

Arguments

d_objectId	The database ID of the object with the reference point parameter.

Value Returned

l_refPointId	Group ID used to record details of the reference point parameter.
nil	Returned if *d_objectId* does not identify an object or there is no reference point parameter defined with it.

Examples

```
refpoint = pcGetRefPointDefn( fig )
```

Retrieves a reference point parameter defined in `fig`.

pcGetRefPoints

```
pcGetRefPoints(
     d_cvId
     )
     => l_refpointlist / nil
```

Description

Returns a list of identifiers for all the reference point parameters in a specified cellview.

You must have defined parameters in this cellview.

Arguments

d_cvId	The database ID of the cellview whose reference point parameters you want listed.

Value Returned

l_refpointlist	List of object IDs used to record details of reference point parameters.
nil	Returned if *d_cvId* does not identify a cellview or if there are no reference point parameters defined in the cellview.

Examples

```
refpointlist = pcGetRefPoints( cv )
```

Gets a list of all the reference point parameters defined in `cv`.

pcGetRepeatDefn

```
pcGetRepeatDefn(
     d_objectId
     )
     => l_repeatlist / nil
```

Description

Returns a list of identifiers for all repetition parameters assigned to an object in the cellview. A single object can be assigned to more than one repetition group.

You must have defined repetition parameters for the cellview.

Arguments

d_objectId	The database ID of the object with repetition parameters.

Value Returned

l_repeatlist	Group ID for the internal structure used to record details of repetition parameters. Returns one object ID for each repeat parameter defined on the given object.
nil	Returned if *d_objectId* does not identify an object or there are no repetition parameters defined with it.

Examples

```
repeat = pcGetRepeatDefn(fig)
```

Retrieves the repetition parameters defined for `fig`.

pcGetRepeats

```
pcGetRepeats(
     d_cvId
     )
     => l_repeatlist / nil
```

Description

Returns a list of identifiers for repetition parameters in the specified cellview.

You must have defined repetition parameters for the cellview.

Arguments

d_cvId	The database ID of the cellview with repetition parameters.

Value Returned

l_repeatlist	List of object IDs used to record the details of repetition parameters.
nil	Returned if *d_cvId* does not identify a cellview or if there are no repetition parameters defined in the cellview.

Examples

```
repeatlist = pcGetRepeats( cv )
```

Lists all repetition parameters defined in `cv`.

pcGetStepDirection

```
pcGetStepDirection(
     l_pointList
     )
     => t_stepDirection / nil
```

Description

Returns the direction from the first point in the list to the second point in the list.

Arguments

l_pointList	List of coordinates in either of the following formats: `'( ( x1 y1 ) ( x2 y2 ) )( x1:y1  x2:y2 )`

Value Returned

t_stepDirection	String stating the direction of the step. Valid Values: `"up"`, `"down"`, `"right"`, `"left"`
`nil`	The direction from the first point in the list to the second point in the list is neither vertical nor horizontal.

Examples

```
pcGetStepDirection( '( 0 0 ) '( 1 0 ) )
=> "right"
```

Returns the direction from point 0:0 to point 1:0.

pcGetSteppedObjectDefn

```
pcGetSteppedObjectDefn(
     d_objectId
     )
     => l_stepobjId / nil
```

Description

Returns an identifier for repetition-along-shape parameters resulting from calls to `pcDefineSteppedObject`. The object cannot be a parameterized shape.

You must have defined the repetition-along-shape parameters for this object.

Arguments

d_objectId	The database ID of the object for which repetition-along-shape parameters are retrieved.

Value Returned

l_stepobjId	List of object IDs used to record details of repetition along parameterized shape parameters.
`nil`	Returned if *d_objectId* does not identify an object that is defined as a parameterized shape.

Examples

```
stepobj = pcGetSteppedObjectDefn( fig )
```

Gets a list of all the repetition-along-shape parameters defined in cellview.

pcGetSteppedObjects

```
pcGetSteppedObjects(
     d_cvId
     )
     => l_stepobjlist / nil
```

Description

Returns a list of identifiers for the repetition-along-shape parameters in the specified cellview.

You must have defined repetition-along-shape parameters for this cellview.

Arguments

d_cvId	The database ID of the cellview containing the repetition-along-shape parameters.

Value Returned

l_stepobjlist	List of object IDs used to record details of the repetition-along-shape parameters.
nil	Returned if *d_cvId* does not identify a cellview or if there is no repetition-along-shape parameters defined in the cellview.

Examples

```
stepobjlist = pcGetSteppedObjects( cv )
```

Gets a list of all the repetition-along-shape parameters defined in `cv`.

pcGetStretchDefn

```
pcGetStretchDefn(
     d_objectId
     )
     => d_StretchId / nil
```

Description

Returns the identifier for a stretch parameter.

You must have defined this line as a stretch control line.

Arguments

d_objectId	The database ID of the line with the stretch parameter.

Value Returned

d_StretchId	Group ID used to record details of the stretch control line.
nil	Returned if a stretch control line is not defined.

Examples

```
stretch = pcGetStretchDefn( stretchline )
```

Retrieves a stretch parameter defined with `stretchline`.

pcGetStretches

```
pcGetStretches(
     d_cvId
     )
     => l_stretchlist / nil
```

Description

Returns a list of identifiers for the stretch parameters in the specified cellview.

You must have defined the stretch control lines.

Arguments

d_cvId	The database ID of the cellview containing the stretch parameters.

Value Returned

l_stretchlist	List of object IDs used to record details of the stretch parameters.
nil	Returned if *d_cvId* does not identify a cellview or if there are no stretch parameters defined in the cellview.

Examples

```
stretchlist = pcGetStretches( cv )
```

Gets a list of all stretch parameters defined in `cv`.

pcGetStretchSummary

```
pcGetStretchSummary(
     d_cvId
     )
     => l_stretchLines / nil
```

Description

For a parameterized master cell created using the Virtuoso Pcell graphical user interface tool, returns a list of lists, where each list contains the field names and values from the Stretch in X and Stretch in Y forms for each stretch line that was defined for the parameterized cell.

Arguments

d_cvId	The database ID of the cellview containing a Pcell master with stretch lines.

Value Returned

l_stretchLines	A list of lists, where each list contains the field names and values from the Stretch in X and Stretch in Y forms for one stretch line that was defined for the Pcell.
nil	The function did not execute successfully.

Examples

```
pcStretchSummary( cv )
```

For a Pcell master containing two stretch lines, returns a list containing two lists, each of which contain the form field names and values for one stretch line; for example:

```
(("Stretch Type: Horizontal " "Name or Expression for Stretch:  Length" "Stretch
Direction:  left" "Reference Dimension (Default):  1.000000")
   ("Stretch Type: Vertical " "Name or Expression for Stretch:  width" "Stretch
Direction:  up" "Reference Dimension (Default):  1.000000")
)
```

pcGrowBox

```
pcGrowBox(
     l_pointList
     xf_margin
     )
     => l_incrementedPointList / nil
```

Description

Increase or decrease the size of the specified box by the specified margin. The system adds *xf_margin* to each coordinate in the list and returns a list of the coordinates of the resulting box.

Arguments

l_pointList	List of coordinates defining the original rectangular box, in either of the following formats:
	`'( ( x1 y1 ) ( x2 y2 ) )`
	`( x1:y1  x2:y2 )`

xf_margin	Positive or negative integer or floating-point number used to increment or decrement the points of the original box. `Valid Values: integer or floating-point number`

Value Returned

l_incrementedPointList

List of coordinates defining the incremented box, in either of the following formats:

```
'( ( x1 y1 ) ( x2 y2 ) )
( x1:y1  x2:y2 )
```

`nil` The function did not complete successfully.

Examples

```
pcGrowBox( '( 0:0 2:2 ) 5.5 )
    =>((-5.5 -5.5)
       (7.5 7.5) )
```

Increments the two points of the original box by 5.5 and returns a list of the points defining the incremented box.

pcGrowPoints

```
pcGrowPoints(
     l_pointList
     xf_margin
     )
     => l_incrementedPointList / nil
```

Description

Increase or decrease the size of a manhattan polygon by the specified margin. The system adds *xf_margin* to each coordinate in the list and returns a list of the coordinates of the resulting polygon.

Arguments

l_pointList	List of coordinates defining the original manhattan polygon, in either of the following formats:
	`'( ( x1 y1 ) ( x2 y2 ) ... ( xn yn ) )`
	`( x1:y1  x2:y2 ... xn:xy )`

xf_margin	Positive or negative integer or floating-point number used to increment or decrement the points of the original manhattan polygon. Valid Values: integer or floating-point number

Value Returned

l_incrementedPointList	List of coordinates defining the incremented manhattan polygon, in either of the following formats:
	`'( ( x1 y1 ) ( x2 y2 ) ... ( xn yn ) )`
	`( x1:y1  x2:y2 ... xn:xy )`

`nil`	The function did not complete successfully.

Examples

```
origPoints = '( 4.0:1.0 5.0:1.0 5.0:2.5 6.5:2.5 6.5:-0.5 4.0:-0.5 )
pcGrowPoints( origPoints 10 )
    =>((-6.0 11.0)
```

```
        (-5.0 11.0)
        (-5.0 12.5)
        (16.5 12.5)
        (16.5 -10.5)
        (-6.0 -10.5)
    )
```

Increments each point of the original manhattan polygon by 10 and returns a list of the points of the incremented polygon.

pcHICompileToSkill

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

Description

Displays the Compile To SKILL form to let you create a SKILL file from the data in the current cellview. The file can then be edited as any SKILL file.

Before you compile a cellview to a SKILL file, you must define all parameters for the cellview.

Arguments

None

Value Returned

`t`	The SKILL file was created.
`nil`	The SKILL file was not created.

pcHIDefineCondition

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

Description

Lets you designate specified objects as conditional by prompting you to select one or more objects in the current cellview. When you complete selecting objects, the system displays the Conditional Inclusion form.

Arguments

None

Value Returned

`t`	The objects were defined as conditional.
`nil`	The objects were not defined as conditional.

pcHIDefineInheritedParameter

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

Description

Lets you designate a Pcell instance whose parameters should be inherited from the Pcell parent in which the instance is placed by prompting you to select the instance in the current cellview. After you select the instance, the system displays the Define/Modify Inherited Parameters form.

Arguments

None

Value Returned

`t`	The function completed successfully.
`nil`	The function did not complete successfully.

pcHIDefineLabel

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

Description

Displays the Define Parameterized Label form to let you create a parameterized label for Pcell you are currently editing. After you complete the form, the system prompts you to enter an anchor point for the label.

Arguments

None

Value Returned

`t`	The label was created.
`nil`	The label was not created.

pcHIDefineLayer

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

Description

Lets you designate specified shapes to be in a parameterized layer group by prompting you to select one or more shapes in the current cellview. When you complete selecting shapes, the system displays the Define Parameterized Layer form.

Arguments

None

Value Returned

`t`	The parameterized label group was created.
`nil`	The parameterized label group was not created.

pcHIDefineParamCell

```
pcHIDefineParamCell(
     [ l_cellIdentifier ]
     )
     => t / nil
```

Description

Displays the Compile To Pcell form to let you create a graphical Pcell (super) master in the database from the design in the current window or from the cellview you specify. If you do not compile a Pcell before you place an instance of it in another design, the system interprets the design as a standard fixed cell instead of a Pcell. Each time you edit a graphical Pcell, you must recompile it so that all placed instances reflect the changes.

You must have already defined parameters for the cellview.

Arguments

l_cellIdentifier	Database ID for the cellview (cellview ID). Default: Current cellview ID

Value Returned

t	The parameterized (super) master was created.
nil	The parameterized (super) master was not created.

Examples

```
pcHIDefineParamCell( cellview_Id )
```

Displays the Compile To Pcell form.

Related Topics

Compile to Pcell

pcHIDefineParameterizedShape

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

Description

Lets you assign the vertices of a shape as parameters of the Pcell you are currently editing by prompting you to select a shape. You can parameterize paths, polygons, and rectangles. When you place an instance of the Pcell, you supply values for the parameters by entering coordinates.

You can define multiple shapes in the same Pcell master as parameterized, as long as they are all of the same shape type. For example, you can define several rectangles as parameterized, but you cannot define one rectangle and one polygon as parameterized in the same Pcell master. For more information about parameterized shapes, see "Parameterized Shapes Commands" in the *Virtuoso Parameterized Cell Reference*.

Arguments

None

Value Returned

`t`	The parameterized shape was created.
`nil`	The parameterized shape was not created.

pcHIDefineParamRefPointObject

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

Description

Lets you specify a reference point parameter as the origin point for a selected object or group of objects in the Pcell you are currently editing. The system prompts you for the reference point, then prompts you to select the object(s). When you complete selecting objects, the system displays the Reference Point by Parameter form. There can be only one reference point parameter defined in a Pcell.

Arguments

None

Value Returned

`t`	The parameterized reference point was created.
`nil`	The parameterized reference point was not created.

pcHIDefinePathRefPointObject

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

Description

Lets you specify the end of a parameterized path as the reference point for objects in the Pcell you are currently editing by prompting you to select the objects. When you complete selecting objects, the system displays the Reference Point by Path Endpoint form. There can be only one reference-point-by-path-endpoint parameter defined in a Pcell.

The parameterized path must exist before you define the reference point.

Arguments

None

Value Returned

`t`	The reference point was created.
`nil`	The reference point was not created.

pcHIDefineProp

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

Description

Displays the Parameterized Property form to let you specify a property for the Pcell you are currently editing.

Arguments

None

Value Returned

`t`	The property was created.
`nil`	The property was not created.

pcHIDefineRepeat

```
pcHIDefineRepeat(
     t_direction
     )
     => d_repeatId
```

Description

Lets you define a repetition parameter for specified objects by prompting you to select one or more objects in the current cellview. When you complete selecting objects, the system displays one of the following forms, depending on the value of the *t_direction* argument: Repeat in X, Repeat in Y, or Repeat in X and Y.

Arguments

t_direction	Direction of repetition. When the value is `2D`, the function creates a two-dimensional array. Valid Values: `horizontal`, `vertical`, `2D`

Value Returned

d_repeatId	Group ID used to store details of the repetition.

Examples

```
pcHIDefineRepeat( "horizontal" )
```

Prompts you to select objects in the current cellview and then displays the Repeat in X form.

pcHIDefineSteppedObject

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

Description

Lets you specify an object or group of objects to repeat along the coordinate string controlling a parameterized shape by prompting you to select the objects in the Pcell you are currently editing. The Pcell must already contain a parameterized shape. When you complete selecting objects, the system displays the Repeat Along Shape form.

Arguments

None

Value Returned

`t`	The repetition group was created.
`nil`	The repetition group was not created.

pcHIDefineStretch

```
pcHIDefineStretch(
     t_direction
     )
     => d_StretchId
```

Description

Allows you to define a stretch parameter for specified objects by prompting you to select one or more objects in the current cellview. The argument *t_direction* determines which form the system displays when you complete selecting objects: Stretch in X or Stretch in Y.

Arguments

t_direction	Direction of the stretch. Valid Values: `right`, `left`, `rightAndLeft`, `up`, `down`, `upAndDown`

Value Returned

d_StretchId	Group ID used to store the details of the stretch parameter.

Examples

```
pcHIDefineStretch( "up" )
```

Prompts you to select objects in the current cellview and then displays the Stretch in Y form.

pcHIDeleteCondition

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

Description

Lets you delete an object from a conditional induction group by prompting you to select the object in the current cellview. When you select the object, the system displays the Delete Conditional Inclusion form.

Arguments

None

Value Returned

`t`	The object was removed from the conditional inclusion group.
`nil`	The object was not removed from the conditional inclusion group.

pcHIDeleteLayer

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

Description

Lets you remove a parameterized layer group by prompting you to select a shape in the group. When you select a shape, the system highlights all objects in the parameterized layer group and displays the Delete Parameterized Layer form.

Arguments

None

Value Returned

`t`	The parameterized layer group was deleted.
`nil`	The parameterized layer group was not deleted.

pcHIDeleteParameterizedShape

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

Description

Lets you delete a parameterized shape from the Pcell you are currently editing by prompting you to select the parameterized shape. When you delete a parameterized shape, the coordinates of the shape are no longer parameters of the Pcell.

Arguments

None

Value Returned

`t`	The parameterized shape was deleted.
`nil`	The parameterized shape was not deleted.

pcHIDeleteProp

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

Description

Displays the Delete Parameterized Property form to let you specify a property to delete from the Pcell you are currently editing. When there is more than one parameterized property defined for the Pcell, click *Next* to view another property.

Arguments

None

Value Returned

`t`	The property was deleted.
`nil`	The property was not deleted.

pcHIDeleteRefPointObject

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

Description

Lets you delete either a reference point defined as a parameter of the cellview or a reference point defined relative to the endpoint of a parameterized path by prompting you to select any object in the reference point group. After you select an object, the system highlights all objects in the group and opens either the Delete Reference Point form or the Delete Reference Point By Path form.

Arguments

None

Value Returned

`t`	The reference point group was deleted.
`nil`	The reference point group was not deleted.

pcHIDeleteRepeat

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

Description

Lets you delete a repeat group from the Pcell you are currently editing by prompting you to select a shape in the repeat group you want to delete. After you select a shape, the system highlights all shapes in the group and displays one of the following forms, depending on the type of repeat group you selected: Delete Repeat in X, Delete Repeat in Y, or Delete Repeat in X and Y.

Arguments

None

Value Returned

`t`	The repeat group was deleted.
`nil`	The repeat group was not deleted.

pcHIDeleteSteppedObject

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

Description

Lets you delete a repetition along shape group from the Pcell you are currently editing by prompting you to select a member (shape) of the repetition along shape group.

Arguments

None

Value Returned

`t`	The repetition along shape group was deleted.
`nil`	The repetition along shape group was not deleted.

pcHIDisplayCondition

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

Description

Highlights a conditional inclusion group in the current cellview window and displays the Show Conditional Inclusion text window with information about the highlighted group. When there is more than one conditional inclusion group, click *OK* in the text window to view the next one.

Arguments

None

Value Returned

`t`	Highlights inclusion groups and displays the Show Conditional Inclusion text window.
`nil`	There are no inclusion groups or the Show Conditional Inclusion text window did not display.

pcHIDisplayInheritedParameter

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

Description

Highlights a Pcell instance whose parameters are inherited from the Pcell parent in which the instance is placed and displays the Show Inherited Parameters text window with information about the highlighted instance. When there is more than one Pcell instance with inherited parameters, click *OK* in the text window to view the next one.

Arguments

None

Value Returned

`t`	Highlights instances with inherited parameters and displays the Show Inherited Parameters text window.
`nil`	There are no instances with inherited parameters or the Show Inherited Parameters text window did not display.

pcHIDisplayLayer

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

Description

Highlights a parameterized layer group in the current cellview window and displays Show Parameterized Layer text window with information about the highlighted group. When there is more than one parameterized layer group, click *OK* in the text window to view the next one.

Arguments

None

Value Returned

`t`	Highlights parameterized layer groups and displays the Show Parameterized Layer text window.
`nil`	There are no parameterized layer groups or the Show Parameterized Layer text window did not display.

pcHIDisplayParameterizedShape

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

Highlights a parameterized shape in the current cellview window and displays the Show Parameterized Shape text window with information about the highlighted group. When there is more than one parameterized shape, click *OK* in the text window to view the next one.

Arguments

None

Value Returned

`t`	Highlights parameterized shapes and displays the Show Parameterized Shape text window.
`nil`	There are no parameterized shapes or the Show Parameterized Shape text window did not display.

pcHIDisplayParams

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

Description

Although the `pcHIDisplayParams` function still displays the *Show Parameters* text window, the information contained in the window might not be complete. The *Show Parameters* command has been replaced by the *Edit Parameters* command. To display information about Pcell parameters, use either the `pcHIEditParameters` function or the `pcHIParamsSummarize` function.

Arguments

None

Value Returned

`t`	The *Show Parameters* text window displays or there are no parameters defined for the Pcell.
`nil`	There are no parameters defined for the Pcell or if the *Show Parameters* text window does not display.

pcHIDisplayProp

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

Description

Displays a Show Parameterized Property text window with information about all parameterized properties defined for the Pcell you are currently editing.

Arguments

None

Value Returned

`t`	The text window was displayed.
`nil`	The text window was not displayed.

pcHIDisplayRefPointObject

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

Description

Highlights a reference point group in the current cellview window and opens the Show Reference Point text window with information about the highlighted group. When there is more than one reference point group, click *OK* in the text window to view the next one.

Arguments

None

Value Returned

`t`	The Show Reference Point text window displays or there are no reference point groups.
`nil`	There are no reference point groups or the Show Reference Point text window did not display.

pcHIDisplayRepeat

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

Highlights a repeat group in the current cellview window and displays one of the following text windows, depending on the type of group you selected: Show Repeat in X, Show Repeat in Y, or Show Repeat in X and Y with information about the highlighted group. When there is more than one repeat group, click *OK* in the text window to view the next one.

Arguments

None

Value Returned

`t`	A repeat group exists and the text window was displayed.
`nil`	A repeat group does not exist.

pcHIDisplaySteppedObject

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

Description

Highlights the objects in a repetition along shape group in the current cellview window and opens the Show Repetition Along Shape text window with information about the highlighted group. When there is more than one repetition along shape group, click *OK* in the text window to view the next one.

Arguments

None

Value Returned

`t`	Highlights objects in a repetition along shape group and opens the text window.
`nil`	There are no repetition along shape groups or the text window does not display.

pcHIEditParameters

```
pcHIEditParameters()
     => t
```

Description

Lets you change the data type and/or value for parameters already defined for the Pcell by displaying the Edit Parameters form.

Arguments

None

Value Returned

`t`	Always returns `t`.

pcHIModifyCondition

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

Description

Lets you add objects to a conditional inclusion group by prompting you to select an object in the inclusion group you want to modify. The system highlights all objects in the selected group and prompts you to select shapes to be added to the group. When you complete selecting the objects, the system displays the Modify Conditional Inclusion form.

Arguments

None

Value Returned

`t`	Selected shapes were added to the conditional inclusion group.
`nil`	Selected shapes were not added to the conditional inclusion group.

pcHIModifyLabel

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

Description

Prompts you to select the parameterized label you want to modify. After you select the label, displays the Modify Parameterized Label form to let you change the values for the selected label.

Arguments

None

Value Returned

`t`	The parameterized label was modified.
`nil`	The parameterized label was not modified.

pcHIModifyLayer

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

Description

Lets you add shapes to a parameterized layer group by prompting you to select an object in the layer group you want to modify. The system highlights all objects in the selected group and prompts you to select shapes to be added to the group. When you complete selecting the shapes, the system displays the Modify Parameterized Layer form.

Arguments

None

Value Returned

`t`	The selected shapes were added to the parameterized layer group.
`nil`	The selected shapes were not added to the parameterized layer group.

pcHIModifyParams

```
pcHIModifyParams()
     => t
```

Description

Lets you change the data type and/or value for parameters already defined for the Pcell by displaying the Edit Parameters form. This function is equivalent to the `pcHIEditParameters` function.

Arguments

None

Value Returned

`t`	Always returns `t`.

pcHIModifyRefPointObject

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

Description

Lets you add objects to a reference group by prompting you to select an object in the group you want to modify. The system highlights all objects in the selected group and prompts you to select shapes to be added to the group. When you complete selecting the shapes, the system lets you change the reference point by displaying either the Reference Point by Parameter form or the Reference Point by Path Endpoint form, depending on the type of group you selected. There can be only one reference point parameter and only one reference point by path endpoint defined for a Pcell.

Arguments

None

Value Returned

`t`	The reference group was modified.
`nil`	The reference group was not modified.

pcHIModifyRepeat

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

Description

Lets you add shapes to a repeat group by prompting you to select a shape in the repeat group you want to modify. After you select an object, the system highlights all shapes in the group and prompts you to select shapes to be added to the group. When you complete selecting shapes, the system displays one of the following forms, depending on the type of repeat group you selected: Modify Repeat in X, Modify Repeat in Y, or Modify Repeat in X and Y.

Arguments

None

Value Returned

`t`	The repeat group was modified.
`nil`	The repeat group was not modified.

pcHIModifySteppedObject

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

Description

Lets you add shapes to a repetition along shape group by prompting you to select a shape in the group you want to modify. After you select a shape, the system highlights all shapes in the group and displays the Modify Repetition Along Shape form.

Arguments

None

Value Returned

`t`	The repetition along shape group was modified.
`nil`	The repetition along shape group was not modified.

pcHIModifyStretchLine

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

Description

Prompts you to select the stretch line you want to modify. After you select a stretch line, displays the Stretch in X or Stretch in Y form, depending on whether you are modifying an X stretch line or a Y stretch line, to let you change the values for the selected stretch line.

Arguments

None

Value Returned

`t`	The stretch line was modified;.
`nil`	The stretch line was not modified.

pcHIQualifyStretchLine

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

Description

Lets you add shapes to be affected by a stretch line by prompting you to select the stretch line. After you select a stretch line, prompts you to select the shapes to be affected. No form is displayed.

Arguments

None

Value Returned

`t`	The stretch line was qualified.
`nil`	The stretch line was not qualified.

pcHIRedefineStretchLine

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

Description

Lets you redefine a previously defined stretch control line or change the parameters assigned to a stretch control line by prompting you to select the stretch line. After you select a stretch line, the system prompts you to draw a stretch line to replace the selected stretch line. After you draw the line, the system displays the Stretch in X or Stretch in Y form, depending on whether you are redefining an X stretch line or a Y stretch line, to let you change the values for the redefined stretch line.

Arguments

None

Value Returned

`t`	The stretch line was redefined.
`nil`	The stretch line was not redefined.

pcHISummarizeParams

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

Description

Displays the Pcell Parameter Summary text window with information about the parameters defined for the Pcell.

Arguments

None

Value Returned

`t`	The Pcell Parameter Summary text box was displayed.
`nil`	The Pcell Parameter Summary text box was not displayed.

pcIsParamSlot

```
pcIsParamSlot(
     g_device
     s_propName
     )
     => t / nil
```

Description

Checks whether the specified `propName` is a parameter slot of the specified device.

Arguments

g_device	A SKILL++ Pcell class object that is inherited from the class `pcParamClass`.
s_propName	A parameter slot name of the given device.

Value Returned

`t`	If the specified `propName` is a parameter slot of the specified device.
`nil`	If the specified `propName` is not a parameter slot of the specified device.

Examples

```
pcell =  makeInstance( 'CORE )
pcIsParamSlot( pcell 'cyanW )
=> t
```

Returns `t` because `cyanW` is defined as a parameter slot in CORE class.

pcModifyParam

```
pcModifyParam(
     d_cvId
     S_param
     t_type
     g_value
     )
     => d_paramId / nil
```

Description

Lets you modify the parameter type and default value for parameters assigned to a compiled Pcell.

Arguments

d_cvId	The database ID of the specified cellview.
S_param	Name of the parameter. Valid Values: a string or symbol
t_type	Type of the parameter. Valid Values: `int`, `float`, `Boolean`, `string`, `ILList`
g_value	Default value of the parameter. Valid Value: any value consistent with the value type specified

Value Returned

d_paramId	Group ID of the property that stores the parameter.
`nil`	Returned if the function does not execute.

Examples

```
pcModifyParam( supermaster "gateWidth" "float" 0.625 )
```

The above example modifies the parameter `gateWidth` to have `float` as its type and `0.625` as its default value.

pcRedefineStretchLine

```
pcRedefineStretchLine(
     d_lineId
     g_paramExpr
     t_direction
     f_defval
     f_minval
     f_maxval
     g_stretchRepeated
     )
     => d_StretchId / nil
```

Description

Redefines the attributes of an existing stretch control line. You can also specify a new location for the stretch control line with this command.

You must have defined the stretch control line.

Arguments

d_lineId	The database ID of the stretch control line.
g_paramExpr	SKILL expression or symbol controlling the stretch.
t_direction	Direction of the stretch. Valid Values: `right`, `left`, `rightAndLeft`, `up`, `down`, `upAndDown`
f_defval	Default value (reference dimension) for the stretch.
f_minval	Minimum value for the stretch.
f_maxval	Maximum value for the stretch. Use `nil` if no maximum is specified.
g_stretchRepeated	Boolean expression indicating whether to stretch shapes repeated in the direction parallel to the stretch.

Value Returned

d_StretchId	Group ID used to store stretch parameter details.
`nil`	Returned if *d_lineId* is not defined as a stretch control line.

Examples

```
pcRedefineStretchLine( stretchLine 'nfetWidth "rightAndLeft" 1.00 1.00 25 nil )
```

Redefines the stretch control line `stretchLine` as controlled by the parameter `nfetWidth`. The stretch direction is `rightAndLeft`. The default and minimum values are `1.00`, and the maximum value is `25`. Horizontally repeated shapes are not affected.

pcRestrictStretchToObjects

```
pcRestrictStretchToObjects(
     d_stretchId
     l_objlist
     )
     => d_stretchId / nil
```

Description

Lets you specify the objects affected by a particular stretch control line. Objects not specified are not moved or stretched by this stretch control line.

Arguments

d_stretchId	The database ID of the stretch control line.
l_objlist	List of objects whose location can be affected by this stretch control line. If `nil`, the stretch applies to all objects.

Value Returned

d_stretchId	Group ID used to store the stretch details.
`nil`	Returned if *d_stretchId* is not defined or if there are no objects in the list of objects.

Examples

```
pcRestrictStretchToObjects( stretchLine
geGetSelSet( getCurrentWindow( ) ) )
```

Restricts the effect of the stretch control line to only the selected objects in the cellview in the current window.

pcRound

```
pcRound(
     n_num
     [ f_precision ]
     [ x_tolerance ]
     )
     => x_result
```

Description

Lets you round a number to the closest integer, using the value of the decimal place specified by *x_tolerance*; additional decimal places are ignored. If the value of the specified decimal place is less than 5, the system drops all decimal places; if the value of the specified decimal place is greater than or equal to 5, the system drops all decimal places and adds one to the integer.

Arguments

n_num	Any number you want to round.
f_precision	Floating-point number specifying number of decimal places to the right of the decimal point to use as a range for determining whether *n_num* is close to a whole number. Default: 0.001
x_tolerance	Positive integer specifying the decimal place to use for rounding. For example, a value of 1 specifies 0.5, 2 specifies 0.05, 3 specifies 0.005, 4 specifies 0.0005, and so on. The numbers in other decimal places are ignored. Default: 1

Value Returned

x_result	An integer. If *n_num* does not contain a number, an error occurs; look in the CIW or at the `CDS.log` file for a message about the error.

Examples

The examples below show numbers rounded using `pcRound`.

```
pcRound( 1.49 )  results in  1
pcRound( -1.49)   results in  -1
pcRound( 1.59 )   results in  2
pcRound( -1.59 )  results in  -2
pcRound( 5.4993 0.001 1 )  results in 6
pcRound( 5.4993 0.001 2 )  results in 5
pcRound( 5.4993 0.002 1 )  results in 6
pcRound( 5.4993 0.0002 1 ) results in 5
```

Related Topics

pcFix

pcExprToString

pcTechFile

Recommended, Supported SKILL Functions for Pcells

pcSetFTermWidth

```
pcSetFTermWidth(
     t_baseName
     x_width
     )
     => t_baseName_width / nil
```

Description

Creates a net or terminal name and assigns a width.

Arguments

t_baseName	String specifying the base name for the net or terminal.
x_width	Integer specifying the width of the net or terminal.

Value Returned

t_baseName_width	String specifying the base name followed by the width, in the following format: `"baseName<n1:n2>"`
`nil`	Returned if the function does not complete successfully.

Examples

```
pcSetFTermWidth( "test" 20 )
    => "test<0:19>"
```

Creates a net or terminal with the base name `"test"` and assigns a width of `20`.

pcSetParamSlotsFromMaster

```
pcSetParamSlotsFromMaster(
     g_device
     d_cv
     )
     => t / nil
```

Description

Sets the class slot values of a specified device to the corresponding Pcell parameter values on a specified Pcell super master or sub master.

Arguments

g_device	An object of the SKILL++ Pcell class, which is inherited from the class `pcParamClass`.
d_cv	A Pcell super master ID or Pcell sub master ID.

Value Returned

`t`	If the class slot values of the specified device are set.
`nil`	Unable to set the device's class slot values.

Examples

```
pcell =  makeInstance( 'CORE )
pcSetParamSlotsFromMaster( pcell pcCellView )
```

In this example, `pcCellView` is either a Pcell super master ID or sub master ID.

Related Topics

pcDefineParamSlot

pcGetDefaultParamsFromClass

pcGetParamSlotType

pcGetParamSlotValue

pcIsParamSlot

pcSetParamSlotsFromMaster

pcSetParamSlotValue

pcSetParamSlotValue

```
pcSetParamSlotValue
     g_device
     s_propName
     g_value
     )
     => g_value
```

Description

Sets the value of the specified parameter slot, `propName`, of the given device.

Arguments

g_device	A SKILL++ Pcell class object that is inherited from class `pcParamClass`.
s_propName	A Pcell slot name of the given device.
g_value	The value of a parameter slot.

Value Returned

g_value	The value of the parameter slot, `propName`, of a given device.

Examples

```
pcell =  makeInstance( 'CORE )
pcSetParamSlotValue( pcell 'cyanW 2.0 )
```

Sets the `cyanW` parameter slot value of the Pcell to 2.0.

pcSkillGen

```
pcSkillGen(
     d_cellViewId
     t_outputFile
     g_isSkillFile
     [ 'disablePrompt ]
     )
     => t / nil
```

Description

Converts a specified cellview into a SKILL file. A SKILL file can be edited and loaded back to a cellview after modification. Loading a SKILL file generates a SKILL master; however, the cellview contains only a label with the text: `Warning: The master is defined by the SKILL procedure associated with the cellview.`

You must have defined the parameters in this cellview.

Arguments

d_cellViewId	Cellview to be converted to SKILL.
t_outputFile	The destination text file.
g_isSkillFile	If `t`, generates a `pcGenCell` procedure. If `nil`, generates a SKILL function.
'disablePrompt	If set to `t`, a prompt, which is displayed during the Pcell master compilation, gets disabled whenever no parameter is specified in the cell.

Value Returned

`t`	The SKILL file was created.
`nil`	The SKILL file was not created.

Examples

```
pcSkillGen(cv ~/mySkillCell t)
```

Puts the SKILL code that creates the cellview `cv` in the `~/mySkillFile` file. For example:

```
let( (  pcMember pcStretchGroup stretchOffsetX stretchOffsetY pcLib
pcMaster pcInst pcTerm pcPin pcPinName
```

```
        pcNet pcTermNet pcNetName pcTermNetName pcMosaicInst
        tpcParameters pcParamProp pcStep pcStepX pcStepY
        pcRepeat pcRepeatX pcRepeatY pcIndexX pcIndexY
        pcLayer pcPurpose pcLabelText pcLabelHeight pcPropText
        pcParamText pcCoords pcPathWidth pcPolygonMargin )
pcLib = pcCellView~>lib
pcParameters = pcCellView~>parameters~>value
; generate all the cv's properties
; --------------------------------


; generate all the primitive shapes in layer "pwell (drawing)"
; -------------------------------------------------------------
pcLayer = 6
pcPurpose = "drawing"
pcInst = dbCreateRect(pcCellView list(pcLayer pcPurpose) list(2:0.5 6.5:7.5))
t

)
pcSkillGen(cv ~/mySkillCell nil)
```

Puts the SKILL code that creates the cellview `cv` inside a `pcGenCell` procedure and writes it to the `~/mySkillFile` file. For example:

```
procedure( pcGenCell( pcCellView "d")
let( (  pcMember pcStretchGroup stretchOffsetX stretchOffsetY pcLib
        pcMaster pcInst pcTerm pcPin pcPinName
        pcNet pcTermNet pcNetName pcTermNetName pcMosaicInst
        tpcParameters pcParamProp pcStep pcStepX pcStepY
        pcRepeat pcRepeatX pcRepeatY pcIndexX pcIndexY
        pcLayer pcPurpose pcLabelText pcLabelHeight pcPropText
        pcParamText pcCoords pcPathWidth pcPolygonMargin )
pcLib = pcCellView~>lib
pcParameters = pcCellView~>parameters~>value

; generate all the cv's properties
; --------------------------------

; generate all the primitive shapes in layer "pwell (drawing)"
; -------------------------------------------------------------
pcLayer = 6
pcPurpose = "drawing"
pcInst = dbCreateRect(pcCellView list(pcLayer pcPurpose) list(2:0.5 6.5:7.5))
t
)
)
```

pcStepAlongShape

```
pcStepAlongShape(
     d_shape1Id
     l_stepDetails(
     l_shape2Points
     )
     => t / nil
```

Description

Replicates a shape (*d_shape1Id*) along a second shape (*l_shape2Points*) by specifying a disembodied property list to define the stepping distance, gap between successive replications, starting offset, ending offset, and object type. The pitch is determined by the offset of the origin of *d_shape1Id* from the point 0:0. You specify the second shape (*l_shape2Points*) with a list of coordinates.

Arguments

d_shape1
: The database ID of the shape to be replicated.

l_stepDetails
: Disembodied property list, which is a list that starts with a SKILL data object, in this case, `nil`, followed by alternating symbol name/value pairs, and is not attached to a specific object or symbol. The attributes contained in the list are:

```
nil
'step          xf_stepValue
'gap           xf_gapValue
'startOffset   xf_startValue
'endOffset     xf_endValue
'objType      t_objTypeValue
```

l_shape2Points
: List of coordinates specifying the second shape, in one of the following formats:

```
list( '( x1 y1 ) '( x2 y2 ) ... '( xn yn ) )
'( x1:y1 x2:y2 ... xn:yn )
```

Value Returned

`t`	The shape, *`d_shape1Id`*, was replicated successfully.
`nil`	The shape, *`d_shape1Id`*, was not replicated successfully.

Examples

```
shape1 = '( ( -2 0 ( ( -2 2 ) ( -1 2 ) ( -1 1 ) ( 0 1 ) ( 0 0 ) )
shape2 = '( ( -14.0 7.5 ) ( 18.5 18.0 ) )
pcStepAlongShape( shape1
    list(
            nil
            'step 4.5
            'gap 1.5
            'startOffset 0
            'endOffset 0
            'objType shape2~>objType
        )
    shape2~>bBox
)
    => t
```

Replicates `shape1`, a polygon, along `shape2`, a rectangle, with a stepping distance of 1.5, gap of 2, starting offset of 0, and ending offset of 2.

Pcell Compiler Customization SKILL Functions

This section provides syntax, descriptions, and examples for the SKILL functions associated with customizing the Pcell compiler.

Related Topics

Customizing the Pcell Compiler

pcUserAdjustParameters

```
pcUserAdjustParameters(
     p_port
     )
     => t / nil
```

Description

A user-defined procedure called by the compiler before it processes any objects. The procedure is normally used to generate code to transform user-specified parameter values, such as to snap them to an even value. Parameters can then be referenced as variables in the SKILL code that is generated.

Arguments

p_port	Port to which the output code is generated.

Value Returned

t | nil	Return value is not relevant.

Examples

```
procedure( pcUserAdjustParameters( port )
; stretch implemented by 2 stretch control lines ⇒
divide parameter value by 2
fprintf(port "ch_width = ch_width/2\n")
)
```

Generates a call to a SKILL procedure to divide the `ch_width` parameter value by `2`.

pcUserGenerateArray

```
pcUserGenerateArray(
     d_mosaic
     t_masterTag
     p_port
     )
     => t / nil
```

Description

A user-defined procedure called by the compiler before it processes any simple arrays (mosaics) in a master Pcell. The procedure is normally used to suppress array generation or to modify arrays.

Arguments

d_mosaic	Database ID of array.
t_masterTag	Name for the master Pcell of the array that can be used in the generated SKILL code.
p_port	Port to which the output code is generated.

Value Returned

t	Compiler does not generate the code to reproduce the array in the submaster Pcell.
nil	Compiler generates the code to reproduce the array in the submaster Pcell as if the procedure were not called.

Examples

```
procedure( pcUserGenerateArray( mosaic master port )
if( mosaic~>name == \"userSpecial\" t nil)
)
```

Suppresses code generation for the array named `userSpecial`.

pcUserGenerateInstance

```
pcUserGenerateInstance(
     d_inst
     t_masterTag
     p_port
     )
     => t / nil
```

Description

A user-defined procedure called by the compiler before it processes any instances in a master Pcell. The procedure is normally used to suppress instance generation or to modify instances.

Arguments

d_inst	Database ID of instance.
t_masterTag	Name for the master Pcell that can be used in the generated SKILL code.
p_port	Port to which the output code is generated.

Value Returned

t	Compiler does not generate the code to reproduce the instance in the submaster Pcell.
nil	Compiler generates the code to reproduce the instance in the submaster Pcell as if the procedure were not called.

Examples

```
procedure( pcUserGenerateInstance( inst master port )
  if( inst~>name == \"userSpecial\" t nil)
)
```

Suppresses code generation for an instance named `userSpecial`.

pcUserGenerateInstancesOfMaster

```
pcUserGenerateInstancesOfMaster(
     d_masterCV
     l_instanceList
     t_tag
     p_port
     )
     => t / nil
```

Description

A user-defined procedure called by the compiler for every master Pcell in a master Pcell. The compiler calls the procedure before it generates code for instances (but not arrays) for the master. The procedure is normally used to generate code to switch masters.

Arguments

d_masterCV	Database ID of master Pcell.
l_instanceList	List of database IDs for all instances of *d_masterCV* in the master Pcell.
t_tag	Name of the master that can be used in SKILL code for placing instances of the master.
p_port	Port to which the output code is generated.

Value Returned

t	Compiler does not generate the code to reproduce the instances of this master Pcell in the submaster Pcell.
nil	Compiler generates code to reproduce the instances of this master Pcell in the submaster Pcell as if the procedure were not called.

Examples

```
procedure( pcUserGenerateInstancesOfMaster(master instances tag port )
if( master~>cellName == \"userSpecialNand\" then
; switch master to generic one
fprintf(port %s = dbOpenCellViewByType(pcLib \"nand\" \"%s\")\n"
tag master~>viewName)
)
; always want code for instances to be generated by
```

```
; compiler
nil
)
```

Replaces instances of `userSpecialNand` cell with instances of `nand` cell.

pcUserGenerateLPP

```
pcUserGenerateLPP(
     d_lpp
     p_port
     )
     => t / nil
```

Description

A user-defined procedure called by the compiler before it processes shapes belonging to layer-purpose pairs in the master Pcell. The procedure is normally used to suppress shape-set generation.

Arguments

d_lpp	Database ID of layer-purpose pair.
p_port	Port to which the output code is generated.

Value Returned

t	Compiler does not generate the code to reproduce any shapes belonging to the layer-purpose pair in the submaster Pcell.
nil	Compiler generates the code to reproduce shapes belonging to the layer-purpose pair in the submaster Pcell as if the procedure were not called.

Examples

```
procedure( pcUserGenerateLPP( lpp port )
if( lpp~>layerName == \"userSpecial\" t nil)
)
```

Suppresses code generation for all shapes on a layer called `userSpecial`.

pcUserGeneratePin

```
pcUserGeneratePin(
     d_pin
     p_port
     )
     => t / nil
```

Description

A user-defined procedure called by the compiler before it processes pins on any terminals in the master Pcell. The procedure is normally used to suppress pin generation or to modify pins.

Arguments

d_pin	Database ID of pin on Pcell.
p_port	Port to which output code is generated.

Value Returned

t	Compiler does not generate code to reproduce the pin in the submaster Pcell.
nil	Compiler generates code to reproduce the pin in the submaster Pcell as if the procedure were not called.

Examples

```
procedure( pcUserGeneratePin( pin port )
    if( pin~>term~>name == \"userSpecial\" t nil )
)
```

Suppresses code generation for the pin if the terminal is called `userSpecial`.

pcUserGenerateProperty

```
pcUserGenerateProperty(
     d_object
     d_prop
     t_tag
     p_port
     )
     => t / nil
```

Description

A user-defined procedure called by the compiler before it processes properties on any objects. The procedure is normally used to suppress property generation in the master Pcell.

Arguments

d_object	The database ID of the object to which the property is attached.
d_prop	The database ID of the property.
t_tag	Name for the object that can be used in any SKILL code generated.
p_port	Port to which the output code is generated.

Value Returned

t	Compiler does not generate code to reproduce the property in the submaster Pcell.
nil	Compiler generates code to reproduce the property in the submaster Pcell as if the procedure were not called.

Examples

```
procedure( pcUserGenerateProperty( obj prop tag port )
if( prop~>name == \"userSpecial\" t nil)
)
```

Suppresses code generation for a property called `userSpecial`.

pcUserGenerateShape

```
pcUserGenerateShape(
     d_shape
     p_port
     )
     => t / nil
```

Description

A user-defined procedure called by the compiler before it processes any shapes in the master Pcell. The procedure is normally used to *suppress* shape generation or to modify shapes.

Arguments

d_shape	Database ID of shape.
p_port	Port to which the output code is generated.

Value Returned

t	Compiler does not generate the code to reproduce the shape.
nil	Compiler generates the code to reproduce the shape as if the procedure were not called.

Examples

```
procedure( pcUserGenerateShape( shape port )
if( cond = shape~>userSpecialProp then
; generate conditional code
fprintf(port "if( %L then\n" cond)
)
; always generate code for this shape
nil
)
```

Looks for the property `userSpecialProp` for the shape. If the property exists, the compiler generates SKILL code to test the results of evaluating the property value when an instance is placed. The code to reproduce the shape is generated by the compiler within a conditional block, so the shape is reproduced in the submaster Pcell only if the condition evaluates to a `nil` value.

You need to close the condition properly (generating closing parentheses) in the call to `pcUserPostProcessObject`.

pcUserGenerateTerminal

```
pcUserGenerateTerminal(
     d_terminal
     p_port
     )
     => t / nil
```

Description

A user-defined procedure called by the compiler before it processes any terminals in the master Pcell. The procedure is normally used to suppress terminal generation or to modify terminals.

Arguments

d_terminal	The database ID of the terminal on the Pcell.
p_port	Port to which the output code is generated.

Value Returned

t	Compiler does not generate code to reproduce the terminal in the submaster Pcell.
nil	Compiler generates code to reproduce the terminal in the submaster Pcell as if the procedure were not called.

Examples

```
procedure( pcUserGenerateTerminal(term port )
    if( term~>name == \"userSpecial\" t nil)
)
```

Suppresses code generation for a terminal called `userSpecial`.

pcUserInitRepeat

```
pcUserInitRepeat(
     l_stepX
     l_stepY
     l_repeatX
     l_repeatY
     p_port
     )
     => t / nil
```

Description

A user-defined procedure called by the compiler before it processes any repetitions. The procedure is normally used to generate code to set the values of variables for repetition parameters.

Arguments

l_stepX	SKILL list for the expression governing the X-stepping distance of the repetition.
l_stepY	SKILL list for the expression governing the Y-stepping distance of the repetition.
l_repeatX	SKILL list for the expression governing the number of repetitions in the X direction.
l_repeatY	SKILL list for the expression governing the number of repetitions in the Y direction.
p_port	Port to which the output code is generated.

Value Returned

t / nil	Return value is not relevant.

Examples

```
procedure( pcUserInitRepeat( sX sY rX rY port )
    ; keep record of step distance and repetitions
    fprintf(port "pcUserStep = %L\n" sY)
    fprintf(port "pcUserRepeat = %L\n" rY)
)
```

Generates a call to SKILL procedures to record repetition parameters.

pcUserPostProcessCellView

```
pcUserPostProcessCellView(
     d_cv
     t_tag
     p_port
     )
     => t / nil
```

Description

A user-defined procedure called by the compiler after it processes any object in a Pcell. The procedure is normally used to generate code to process a list of objects that was built during compilation.

Arguments

d_cv	The database ID of the master Pcell being processed.
t_tag	Name that can be used to refer to the Pcell in the SKILL code output by the procedure.
p_port	Port to which the output code is generated.

Value Returned

t | nil	Return value is not relevant.

Examples

```
procedure( pcUserPostProcessCellView( cv tag port )
    ; adjust contacts up by half "slop" amount
fprintf(port "foreach( contact PCUserContacts \n")
fprintf(port "    dbMoveShape(contact pcCellView
list( 0:(width - PCUserRepeat*PCUserStep) / 2
\"R0\"))\n")
fprintf(port ")\n")
)
```

Generates a call to a SKILL procedure to move the list of database objects.

pcUserPostProcessObject

```
pcUserPostProcessObject(
     d_obj
     t_tag
     p_port
     )
     => t / nil
```

Description

A user-defined procedure called by the compiler after it processes any object (instance, shape, terminal, and so forth) in a master Pcell. The procedure is normally used to generate code to modify a generated object.

Arguments

d_obj	Database ID of object.
t_tag	Name for the object in the generated SKILL code.
p_port	Port to which the output code is generated.

Value Returned

`t / nil`	Return value is not relevant.

Examples

```
procedure( pcUserPostProcessObject(obj tag port )
if( obj~>objType == "inst"
&& obj~>userSpecialProp then
    ; generate code to change its magnification
fprintf(port "%s~>mag = pcUserMagScale\n" tag)
)
)
```

Checks to see if the object is an instance and has a property `userSpecialProp`. If so, code is generated to change the magnification of the instance to `pcUserMagScale` (which was defined using `pcUserPreProcessCellView`).

pcUserPreProcessCellView

```
pcUserPreProcessCellView(
     d_cv
     t_tag
     p_port
     )
     => t / nil
```

Description

A user-defined procedure called by the compiler before it processes any objects in a Pcell. The procedure is normally used to generate code to initialize variables before the compiler processes individual objects.

Arguments

d_cv	The database ID of the master Pcell being processed.
t_tag	Name that can be used to refer to the Pcell in the SKILL code output by the procedure.
p_port	Port to which the output code is generated.

Value Returned

t / nil	Return value is not relevant.

Examples

```
procedure(pcUserPreProcessCellView( cv tag port )
    ; Initialize list to store the contacts
fprintf(port "PCUserContacts = nil\n")
)
```

Generates a call to a SKILL procedure to initialize a list of database objects.

pcUserSetTermNetName

```
pcUserSetTermNetName(
     d_pinFig
     p_port
     )
     => t / nil
```

Description

A user-defined procedure called by the compiler before it processes any pins on any terminals that are part of a repetition group in the master Pcell. The procedure is normally used to customize the connectivity of replicated pins. SKILL code generated by this procedure should assign the net name to the SKILL variable `pcTermNetName`. This is the net name used in the code generated by the compiler to create terminals in the submaster Pcell. The SKILL variables `pcIndexX` and `pcIndexY` are available for incorporation into `pcTermNetName` if you need to make different nets for each different repeated pin.

Arguments

d_pinFig	The database ID of the figure associated with the pin.
p_port	Port to which the output code is generated.

Value Returned

`t`	Compiler does not generate code to define the terminal net name in the submaster Pcell.
`nil`	Compiler generates code to name the terminal net in the submaster Pcell as if the procedure were not called.

Examples

```
procedure( pcUserSetTermNetName( fig port )
if( fig~>pin~>term~>name == \"userSpecial\" then
fprintf( port "pcTermNetName = get_string( concat(
\"userSpecial\" pcIndexX ) ) \n" )
t
else ; let compiler generate net name
nil
)
)
```

Generates code to define a net name for the terminal.

Parameterized Cell SKILL Cross-Reference Table

This table summarizes the procedural and interactive `pc` SKILL functions associated with Parameterized Cell Compiler (Pcell) menu commands. A complete listing of the syntax, descriptions, and examples for the Pcell SKILL functions follows this table.

Interactive Function	Procedural Function	Menu Command
pcHIDefineParamCell	pcDefinePCell	*Compile – To Pcell*
pcHICompileToSkill	pcSkillGen	*Compile – To Skill File*
pcHIDefineCondition	pcDefineCondition pcGetConditions pcGetParameters	*Conditional Inclusion – Define*
pcHIDeleteCondition	pcDeleteCondition pcDefineCondition pcGetConditions	*Conditional Inclusion – Delete*
pcHIModifyCondition	Not available	*Conditional Inclusion – Modify*
pcHIDisplayCondition	Not available	*Conditional Inclusion – Show*
pcHIDefineInheritedParameter	pcDefineInheritParam pcGetInheritParamDefn pcGetParameters	*Inherited Parameters – Define/Modify*
pcHIDisplayInheritedParameter	No procedural SKILL function	*Inherited Parameters – Show*
pcHIDefineLabel	pcDefineParamLabel pcGetParameters pcGetParamLabelDefn pcGetParamLabels	*Parameterized Label – Define*
pcHIModifyLabel	No procedural function	*Parameterized Label – Modify*

Interactive Function	Procedural Function	Menu Command
pcHIDefineLayer	pcDefineParamLayer pcGetParameters pcGetParamLayers pcGetParamLayerDefn	*Parameterized Layer – Define*
pcHIDeleteLayer	pcDeleteParamLayer pcDefineParamLayer	*Parameterized Layer – Delete*
pcHIModifyLayer	No procedural function	*Parameterized Layer – Modify*
pcHIDisplayLayer	No procedural function	*Parameterized Layer – Show*
pcHIDefineProp	pcDefineParamProp pcGetParameters pcGetParamProps	*Parameterized Property – Define/Modify*
pcHIDeleteProp	pcDeleteParamProp	*Parameterized Property – Delete*
pcHIDisplayProp	No procedural function	*Parameterized Property – Show*
pcHIDefineParameterizedShape	pcDefineParamPolygon pcDefineParamPath pcDefineParamRect pcGetParameters pcGetParamShapeDefn pcGetParamShapes	*Parameterized Shapes – Define/Modify*
pcHIDeleteParameterizedShape	pcDeleteParamShape	*Parameterized Shapes – Delete*
pcHIDisplayParameterizedShape	No procedural function	*Parameterized Shapes – Show*
pcHIEditParameters	pcModifyParam	*Parameters – Edit Parameters*

Interactive Function	Procedural Function	Menu Command
pcHIModifyParams	pcModifyParam	*Parameters – Edit Parameters*
pcHIDisplayParams	No procedural function	*Parameters – Show*
pcHISummarizeParams	No procedural function	*Parameters – Summarize*
pcHIDefineParamRefPointObject	pcDefineParamRefPointObject pcGetParameters pcGetRefPointDefn pcGetRefPoints	*Reference Point – Define by Parameter*
pcHIDefinePathRefPointObject	pcDefinePathRefPointObject pcGetParameters pcGetRefPointDefn pcGetRefPoints	*Reference Point – Define by Path Endpoint*
pcHIDeleteRefPointObject	pcDeleteRefPoint pcDefinePathRefPointObject pcDefineParamRefPointObject	*Reference Point – Delete*
pcHIModifyRefPointObject	No procedural function	*Reference Point – Modify*
pcHIDisplayRefPointObject	No procedural function	*Reference Point – Show*
pcHIDeleteRepeat	pcDeleteRepeat pcDefineRepeat pcGetRepeats	*Repetition – Delete*
pcHIModifyRepeat	No procedural function	*Repetition – Modify*

Interactive Function	Procedural Function	Menu Command
pcHIDefineRepeat	pcDefineRepeat	*Repetition*
	pcGetParameters	*– Repeat in X*
	pcGetRepeatDefn	*– Repeat in Y*
	pcGetRepeats	*– Repeat in X and Y*
pcHIDisplayRepeat	No procedural function	*Repetition – Show*
pcHIDefineSteppedObject	pcDefineSteppedObject	*Repetition Along Shape – Define*
	pcGetParameters	
	pcGetSteppedObjectDefn	
	pcGetSteppedObjects	
pcHIDeleteSteppedObject	pcDeleteSteppedObject	*Repetition Along Shape – Delete*
	pcDefineSteppedObject	
pcHIModifySteppedObject	No procedural function	*Repetition Along Shape – Modify*
pcHIDisplaySteppedObject	No procedural function	*Repetition Along Shape – Show*
pcHIModifyStretchLine	pcRedefineStretchLine	*Stretch – Modify*
pcHIQualifyStretchLine	pcRestrictStretchToObjects	*Stretch – Qualify*
pcHIRedefineStretchLine	No procedural function	*Stretch – Redefine*
	pcGetStretchSummary	*Parameters – Summarize*
pcHIDefineStretch	pcDefineStretchLine	*Stretch*
	pcGetParameters	*– Stretch in X*
	pcGetStretchDefn	*– Stretch in Y*
	pcGetStretches	
	pcRedefineStretchLine	
	pcGetOffsetPath	
	pcGetOffsetPolygon	

Interactive Function	Procedural Function	Menu Command
—	—	*Connectivity – Nets – Add Shape*
—	—	*Connectivity – Nets – Remove Shape*
auHiUltraPCell	No procedural function	*Make Ultra Pcell*

3

Express Pcells Data Management Functions

This topic describes SKILL functions for Express Pcells data management. These SKILL functions operate only when the Express Pcell feature is enabled by setting the `CDS_ENABLE_EXP_PCELL` environment variable to `true`.

Related Topics

Express Pcells

dbClearPcellCache

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

Description

Deletes all the submasters of the cellview specified by *t_libName*, *t_cellName*, and *t_viewName* from the Express Pcell cache on disk at the location specified by the CDS_EXP_PCELL_DIR environment variable. If no argument is specified then it deletes the complete cache.

Arguments

t_libName	Specifies the name of the library.
t_cellName	Specifies the name of the cell.
t_viewName	Specifies the name of the view.

Value Returned

t	All submasters of the specified cellview are deleted from the cache.
nil	Unable to delete submasters from the cache.

Related Topics

Express Pcells

Maintaining the PDK Version in the Express Pcell Cache

dbSavePcellCache

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

Description

Saves the submasters that exist in virtual memory of the current Virtuoso session to the Express Pcell cache on disk at the location specified by the `CDS_EXP_PCELL_DIR` environment variable.

If there is a mismatch in the timestamps of the supermaster, the existing submasters in the cache are deleted, and only the new submasters generated in virtual memory are saved to the Express Pcell cache on disk.

Note: This SKILL function does not does not support schematic (and symbol) Pcells. For more information, contact Cadence Support.

Arguments

None

Value Returned

`t`	All submasters in the virtual memory are saved to the cache.
`nil`	Unable to save the submasters to the cache.

Related Topics

Express Pcells

Maintaining the PDK Version in the Express Pcell Cache

dbSavePcellCacheForCV

```
dbSavePcellCacheForCV(
     t_libName
     t_cellName
     t_viewName
     [ n_openLevels ]
     )
     => t / nil
```

Description

Opens the cellview specified by *t_libName*, *t_cellName*, and *t_viewName* and saves all the existing submasters generated or updated in virtual memory to the Express Pcell cache (including any other pre-existing submasters in virtual memory) on disk at the location specified by the CDS_EXP_PCELL_DIR environment variable.

If there is a mismatch in the timestamps of the supermaster, the existing submasters in the cache are deleted, and only the new submasters generated in virtual memory are saved to the Express Pcell cache on disk.

This SKILL function does not does not support schematic (and symbol) Pcells. For more information, contact Cadence Support.

Arguments

t_libName	Specifies the name of the library
t_cellName	Specifies the name of the cell
t_viewName	Specifies the name of the view
n_openLevels	Is an optional argument. The default value of this argument is 32. Therefore, the complete cellview hierarchy opens by default.

Value Returned

t	Opens the specified cellview and saves the existing submasters generated or updated in the virtual memory to the Express Pcell cache.
nil	The operation failed.

Related Topics

Express Pcells

Maintaining the PDK Version in the Express Pcell Cache

dbSavePcellCacheForCVOnly

```
dbSavePcellCacheForCVOnly(
     t_libName
     t_cellName
     t_viewName
     [ n_depth ]
     )
     => t / nil
```

Description

Opens the cellview specified by *t_libName*, *t_cellName*, and *t_viewName* and saves all the Pcell submasters specified in the given cell in the Express Pcell Cache.

Note: This SKILL function does not does not support schematic (and symbol) Pcells. For more information, contact Cadence Support.

Arguments

t_libName	Name of the library.
t_cellName	Name of the cell.
t_viewName	Name of the view.
n_depth	An optional argument. It specifies the level in the hierarchy of the given lib/cell/view till where the instantiated Pcell submaster should be saved. If not specified, then the command saves all the submasters including the leaf level.

Value Returned

t	Successfully saved all the submasters to the cache.
nil	Unable to save the submasters to the cache.

Examples

```
dbSavePcellCacheForCVOnly( lib cell view 3 )
```

Saves all the Pcell submasters in the lib/cell/view hierarchy till the depth of 3.

Related Topics

Express Pcells

Maintaining the PDK Version in the Express Pcell Cache

xpcDumpCache

```
xpcDumpCache(
     g_detailReport
     [ t_filename ]
     )
```

Description

This SKILL API dumps the Express Pcell cache information to CIW and in the specified file.

When using this SKILL function ensure that environment variables, `CDS_ENABLE_EXP_PCELL` and `CDS_EXP_PCELL_DIR` are set appropriately.

Arguments

g_detailReport	When set to `nil`, generates a short report that lists all supermasters represented in cache, followed by the number of submasters (stored in the cache) for each supermaster Whens set to `t`, generates a detailed report that in addition to the information provided in the short report, lists each submaster in the cache with the list of parameter names and values.
t_filename	If a file name is specified and it is in write mode, the cache information is dumped to both CIW and in the specified file. Otherwise, the cache information is dumped only in CIW. The value that you specify for this argument should be enclosed in quotation marks.

Value Returned

t	Successfully dumps the Express Pcell cache information to CIW and in the specified file.
nil	Unable to save the submasters to the cache.

Examples

Short report dumped to CIW and specified file

```
xpcDumpCache(
nil "file1.dump"
)
```

Short report of the cache information is dumped to both CIW and in the `file1.dump` file.

```
Express PCell Cache Dump Report:
--------------------------------
SuperMaster(lib/cell/view)       Number of submasters
PDKLIB/p/layout                         1

PDKLIB/n/layout                         1
```

Short Report dumped only to CIW

```
xpcDumpCache( nil )
```

Short report of the cache information is dumped only to CIW.

```
Express PCell Cache Dump Report:
--------------------------------
SuperMaster(lib/cell/view)              Number of submasters
testLib/customCell/layout                      5
```

Detailed report dumped to both CIW and the specified file

```
xpcDumpCache(
t "file2.dump"
)
```

Detailed report of the cache information is dumped to both CIW and in the `file2.dump` file.

```
Express PCell Cache Dump Report:
--------------------------------
SuperMaster(lib/cell/view)                                  Number of submasters

PDKLIB/p/layout                                             1

Parameter values (<Parameter Name> = <Parameter Value>) are:-

cntOffsetR=0.000000   drainStrap=none  drawDNW=0      drawGuard=1      drawWell=1

rmRightSD=0           sa=130n          sb=130n        sc=400n          sd=0.14u

sourceStrap=none      strap=0          strapGate=none strapGateTop=1   swapSD=0

topStrapExt=0.0       wf=1u
```

```
PDKLIB/n/layout       1

Parameter values (<Parameter Name> = <Parameter Value>) are:-

drawGuard=1    drawWell=1    dualStrapGate=0  enableMfactor=0   extendLeftGate=0

sc=400n        sd=0.14u      strap=0          strapGate=none    strapGateTop=1
```

dbUpdatePcellCache

```
dbUpdatePcellCache(
     [ t_libName ]
     [ t_cellName ]
     [ t_viewName ]
     [ g_checkTimeStamp ]
     )
     => t / nil
```

Description

Updates and saves the submasters that exist in cache by re-evaluating the Pcells. If Library, cell, and view is specified, it updates the submasters only for the specified supermaster.

It does not consider any existing submaster in virtual memory.

If the number or type of Pcell parameters get changed, all the submasters corresponding to that Pcell supermaster are deleted from the cache.

This SKILL function does not does not support schematic (and symbol) Pcells. For more information, contact Cadence Support.

Arguments

t_libName	Specifies the name of the library.
t_cellName	Specifies the name of the cell.
t_viewName	Specifies the name of the view.
g_checkTimeStamp	Whether to check the time stamp for updating the cache. If set to `true`, then it will update only for those super masters whose time stamp have been changed. Else, it will update for all the supermasters. Default: `False`.

Related Topics

Express Pcells

Maintaining the PDK Version in the Express Pcell Cache

xpcEnableExpressPcell

```
xpcEnableExpressPcell(
     g_enable
     )
```

Description

Enables or disables the Express Pcell cache.

Argument

g_enable	When `t`, the express Pcell cache is enabled irrespective of the previous cache state. When `nil`, the express Pcell cache is disabled irrespective of the previous cache state.

Related Topics

Express Pcell

xpcGetPDKVersion

```
xpcGetPDKVersion(
     [ t_libName ]
     )
     => t_libVersion / l_libVersion / nil
```

Description

Returns the version of the library or libraries stored in the Express Pcell cache.

Arguments

t_libName	Name of a library in the Express Pcell cache.

Value Returned

t_libVersion	Version of the specified library.
l_libVersion	List of the versions of all the libraries present in the Express Pcell cache. This list is displayed if the library name is not specified.
nil	The operation failed. If the specified library is not available, a message is displayed to indicate that the version is unavailable.

Examples

Lists the versions of all the libraries present in the Express Pcell cache.

```
xpcGetPDKVersion()
 (("testLib1" "5.0")
     ("testLib2" "")
 )
```

Lists version of the Express Pcell library `testLib1`.

```
xpcGetPDKVersion("testLib1")
 ("5.0")
```

Version cannot be detected because the specified library is not present in the Express Pcell cache.

```
xpcGetPDKVersion("testLib3")
("Version Not Available")
```

Related Topics

Express Pcell