

# **Virtuoso Parameterized Cell SKILL Reference**

**Product Version ICADVM20.1  
October 2020**

© 2020 Cadence Design Systems, Inc. All rights reserved.  
Printed in the United States of America.

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

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

**Trademarks:** Trademarks and service marks of Cadence Design Systems, Inc. contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522. All other trademarks are the property of their respective holders.

**Restricted Permission:** This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used only in accordance with a written agreement between Cadence and its customer.
2. The publication may not be modified in any way.
3. Any authorized copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
4. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

**Disclaimer:** Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

**Restricted Rights:** Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor.

---

# Contents

---

<u>Preface</u> .....	9
<u>Scope</u> .....	10
<u>Licensing Requirements</u> .....	10
<u>Related Documentation</u> .....	10
<u>What's New and KPNS</u> .....	10
<u>Installation, Environment, and Infrastructure</u> .....	10
<u>Technology Information</u> .....	11
<u>Virtuoso Tools</u> .....	11
<u>SKILL Documents</u> .....	11
<u>Additional Learning Resources</u> .....	12
<u>Video Library</u> .....	12
<u>Virtuoso Videos Book</u> .....	12
<u>Rapid Adoption Kits</u> .....	12
<u>Help and Support Facilities</u> .....	12
<u>Customer Support</u> .....	13
<u>Feedback about Documentation</u> .....	13
<u>Understanding Cadence SKILL</u> .....	15
<u>Using SKILL Code Examples</u> .....	15
<u>Sample SKILL Code</u> .....	15
<u>Accessing API Help</u> .....	16
<u>Typographic and Syntax Conventions</u> .....	17
<u>Identifiers Used to Denote Data Types</u> .....	18

## 1

<u>Parameterized Cell Functions</u> .....	21
<u>Safety Rules for Creating SKILL Pcells</u> .....	22
<u>Calling SKILL Procedures from within Pcells</u> .....	23
<u>Using Macros in Pcell SKILL Code</u> .....	23
<u>Physical Limits for Functions</u> .....	23
<u>Recommended, Supported SKILL Functions for Pcells</u> .....	24
<u>Finding Supported SKILL Functions</u> .....	25

## Virtuoso Parameterized Cell SKILL Reference

---

<u>Using print Functions in a Pcell</u>	26
<u>Enclosing the Body of Code in a let or prog for Local Variables</u>	26
<u>What to Avoid When Creating Pcells</u>	27
<u>About Pcell Super and Submaster Cells</u>	29
<u>Using the SKILL Operator ~&gt; with Pcells</u>	29
<u>pc Functions for SKILL Pcell Code</u>	31
<u>pcExprToString</u>	32
<u>pcTechFile</u>	33

## 2

<u>Graphical Parameterized Cell Functions</u>	1
<u>Graphical Pcell Functions</u>	2
<u>pcColinearPoints</u>	2
<u>pcConcatOrient</u>	4
<u>pcDefineCondition</u>	5
<u>pcDefineInheritParam</u>	7
<u>pcDefineParamCell</u>	8
<u>pcDefineParamLabel</u>	9
<u>pcDefineParamLayer</u>	10
<u>pcDefineParamPath</u>	12
<u>pcDefineParamPolygon</u>	14
<u>pcDefineParamProp</u>	16
<u>pcDefineParamRect</u>	17
<u>pcDefineParamRefPointObject</u>	18
<u>pcDefinePathRefPointObject</u>	19
<u>pcDefinePCell</u>	20
<u>pcDefineParamSlot</u>	24
<u>pcDefinePPCell</u>	25
<u>pcDefineRepeat</u>	31
<u>pcDefineSteppedObject</u>	34
<u>pcDefineStretchLine</u>	36
<u>pcDeleteCondition</u>	38
<u>pcDeleteParam</u>	39
<u>pcDeleteParamLayer</u>	40
<u>pcDeleteParamProp</u>	41

## Virtuoso Parameterized Cell SKILL Reference

---

<u>pcDeleteParamShape</u>	42
<u>pcDeleteRefPoint</u>	43
<u>pcDeleteRepeat</u>	44
<u>pcDeleteSteppedObject</u>	45
<u>pcDraw</u>	46
<u>pcExprToProp</u>	48
<u>pcFilterPoints</u>	49
<u>pcFix</u>	50
<u>pcGetBendAngle</u>	53
<u>pcGetCodeParamNames</u>	54
<u>pcGetCodeParamValue</u>	55
<u>pcGetConditions</u>	56
<u>pcGetDefaultParamsFromClass</u>	57
<u>pcGetInheritParamDefn</u>	58
<u>pcGetInheritParams</u>	59
<u>pcGetOffsetPath</u>	60
<u>pcGetOffsetPolygon</u>	62
<u>pcGetParamSlotType</u>	64
<u>pcGetPathRefPoint</u>	65
<u>pcGetParameters</u>	68
<u>pcGetParamLabelDefn</u>	69
<u>pcGetParamLabels</u>	70
<u>pcGetParamLayers</u>	71
<u>pcGetParamLayerDefn</u>	72
<u>pcGetParamProps</u>	73
<u>pcGetParamShapeDefn</u>	74
<u>pcGetParamShapes</u>	75
<u>pcGetParamSlotValue</u>	76
<u>pcGetRefPointDefn</u>	77
<u>pcGetRefPoints</u>	78
<u>pcGetRepeatDefn</u>	79
<u>pcGetRepeats</u>	80
<u>pcGetStepDirection</u>	81
<u>pcGetSteppedObjectDefn</u>	82
<u>pcGetSteppedObjects</u>	83
<u>pcGetStretchDefn</u>	84

## Virtuoso Parameterized Cell SKILL Reference

---

<u>pcGetStretches</u>	85
<u>pcGetStretchSummary</u>	86
<u>pcGrowBox</u>	87
<u>pcGrowPoints</u>	89
<u>pcHICompileToSkill</u>	91
<u>pcHIDefineCondition</u>	92
<u>pcHIDefineInheritedParameter</u>	93
<u>pcHIDefineLabel</u>	94
<u>pcHIDefineLayer</u>	95
<u>pcHIDefineParamCell</u>	96
<u>pcHIDefineParameterizedShape</u>	97
<u>pcHIDefineParamRefPointObject</u>	98
<u>pcHIDefinePathRefPointObject</u>	99
<u>pcHIDefineProp</u>	100
<u>pcHIDefineRepeat</u>	101
<u>pcHIDefineSteppedObject</u>	102
<u>pcHIDefineStretch</u>	103
<u>pcHIDeleteCondition</u>	104
<u>pcHIDeleteLayer</u>	105
<u>pcHIDeleteParameterizedShape</u>	106
<u>pcHIDeleteProp</u>	107
<u>pcHIDeleteRefPointObject</u>	108
<u>pcHIDeleteRepeat</u>	109
<u>pcHIDeleteSteppedObject</u>	110
<u>pcHIDisplayCondition</u>	111
<u>pcHIDisplayInheritedParameter</u>	112
<u>pcHIDisplayLayer</u>	113
<u>pcHIDisplayParameterizedShape</u>	114
<u>pcHIDisplayParams</u>	115
<u>pcHIDisplayProp</u>	116
<u>pcHIDisplayRefPointObject</u>	117
<u>pcHIDisplayRepeat</u>	118
<u>pcHIDisplaySteppedObject</u>	119
<u>pcHIEditParameters</u>	120
<u>pcHIModifyCondition</u>	121
<u>pcHIModifyLabel</u>	122

## Virtuoso Parameterized Cell SKILL Reference

---

<u>pcHIModifyLayer</u>	123
<u>pcHIModifyParams</u>	124
<u>pcHIModifyRefPointObject</u>	125
<u>pcHIModifyRepeat</u>	126
<u>pcHIModifySteppedObject</u>	127
<u>pcHIModifyStretchLine</u>	128
<u>pcHIQualifyStretchLine</u>	129
<u>pcHIRedefineStretchLine</u>	130
<u>pcHISummarizeParams</u>	131
<u>pclsParamSlot</u>	132
<u>pcModifyParam</u>	133
<u>pcRedefineStretchLine</u>	134
<u>pcRestrictStretchToObjects</u>	136
<u>pcRound</u>	137
<u>pcSetFTermWidth</u>	139
<u>pcSetParamSlotsFromMaster</u>	140
<u>pcSetParamSlotValue</u>	141
<u>pcSkillGen</u>	142
<u>pcStepAlongShape</u>	144
<u>auHiUltraPCell</u>	146
<u>Pcell Compiler Customization SKILL Functions</u>	148
<u>pcUserAdjustParameters</u>	149
<u>pcUserGenerateArray</u>	150
<u>pcUserGenerateInstance</u>	151
<u>pcUserGenerateInstancesOfMaster</u>	152
<u>pcUserGenerateLPP</u>	154
<u>pcUserGeneratePin</u>	155
<u>pcUserGenerateProperty</u>	156
<u>pcUserGenerateShape</u>	157
<u>pcUserGenerateTerminal</u>	158
<u>pcUserInitRepeat</u>	159
<u>pcUserPostProcessCellView</u>	160
<u>pcUserPostProcessObject</u>	161
<u>pcUserPreProcessCellView</u>	162
<u>pcUserSetTermNetName</u>	163
<u>Parameterized Cell SKILL Cross-Reference Table</u>	164

### 3

<u>Express Pcells Data Management Functions</u> .....	169
<u>dbClearPcellCache</u> .....	170
<u>dbSavePcellCache</u> .....	171
<u>dbSavePcellCacheForCV</u> .....	172
<u>dbSavePcellCacheForCVOnly</u> .....	173
<u>dbUpdatePcellCache</u> .....	174
<u>xpcEnableExpressPcell</u> .....	175
<u>xpcDumpCache</u> .....	176



# Preface

---

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

This preface contains the following topics:

- [Scope](#)
- [Licensing Requirements](#)
- [Related Documentation](#)
- [Additional Learning Resources](#)
- [Customer Support](#)
- [Feedback about Documentation](#)
- [Understanding Cadence SKILL](#)
- [Typographic and Syntax Conventions](#)

- [Identifiers Used to Denote Data Types](#)

## Scope

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

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

## Licensing Requirements

For information on licensing in the Virtuoso design environment, see the [\*Virtuoso Software Licensing and Configuration Guide\*](#).

## Related Documentation

### What's New and KPNS

- [\*Virtuoso Parameterized Cell What's New\*](#)
- [\*Virtuoso Parameterized Cell Known Problems and Solutions\*](#)

### Installation, Environment, and Infrastructure

- [\*Virtuoso Parameterized Cell SKILL Reference\*](#)
- [\*Virtuoso Relative Object Design User Guide\*](#)
- [\*Sample Parameterized Cells Installation and Reference\*](#)
- [\*Cadence Installation Guide\*](#)

# Virtuoso Parameterized Cell SKILL Reference

## Preface

---

- [\*Virtuoso Design Environment User Guide\*](#)
- [\*Cadence Application Infrastructure User Guide\*](#)
- [\*Component Description Format User Guide\*](#)

## Technology Information

- [\*Virtuoso Technology Data User Guide\*](#)
- [\*Virtuoso Technology Data ASCII Files Reference\*](#)
- [\*Virtuoso Technology Data SKILL Reference\*](#)

## Virtuoso Tools

- [\*Virtuoso Layout Suite SKILL Reference\*](#)
- [\*Virtuoso Layout Suite XL User Guide\*](#)
- [\*Virtuoso Schematic Editor L User Guide\*](#)
- [\*Virtuoso Space-based Router User Guide\*](#)
- [\*Virtuoso Design Rule Driven Editing User Guide\*](#)
- [\*Virtuoso Relative Object Design User Guide\*](#)
- [\*Design Data Translator's Reference\*](#)

## SKILL Documents

- [\*Virtuoso Design Environment SKILL Reference\*](#)
- [\*Cadence SKILL Language User Guide\*](#)
- [\*Cadence SKILL Language Reference\*](#)
- [\*Cadence SKILL Development Reference\*](#)
- [\*Virtuoso Technology Data SKILL Reference\*](#)
- [\*Virtuoso Layout Suite SKILL Reference\*](#)
- [\*Virtuoso Schematic Editor SKILL Reference\*](#)
- [\*Cadence User Interface SKILL Reference\*](#)

■ [Cadence Interprocess Communication SKILL Reference](#)

## Additional Learning Resources

### Video Library

The [Video Library](#) on the Cadence Online Support website provides a comprehensive list of videos on various Cadence products.

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

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

### Virtuoso Videos Book

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

### Rapid Adoption Kits

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

To explore the full range of training courses provided by Cadence in your region, visit [Cadence Training](#) or write to [training\\_enroll@cadence.com](mailto:training_enroll@cadence.com).

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

### Help and Support Facilities

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

- The Virtuoso *Help* menu provides consistent help system access across Virtuoso tools and applications. The standard Virtuoso *Help* menu lets you access the most useful help

and support resources from the Cadence support and corporate websites directly from the CIW or any Virtuoso application.

- The Virtuoso Welcome Page is a self-help launch pad offering access to a host of useful knowledge resources, including quick links to content available within the Virtuoso installation as well as to other popular online content.

The Welcome Page is displayed by default when you open Cadence Help in standalone mode from a Virtuoso installation. You can also access it at any time by selecting *Help – Virtuoso Documentation Library* from any application window, or by clicking the *Home* button on the Cadence Help toolbar (provided you have not set a custom home page).

For more information, see [Getting Help](#) in *Virtuoso Design Environment User Guide*.

## Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

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

- Log on to Cadence Online Support

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

## Feedback about Documentation

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

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

You can also submit feedback by using the following methods:

- In the Cadence Help window, click the *Feedback* button and follow instructions.

# Virtuoso Parameterized Cell SKILL Reference

## Preface

---

- On the Cadence Online Support [Product Manuals](#) page, select the required product and submit your feedback by using the *Provide Feedback* box.

## Understanding Cadence SKILL

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

For more information about the SKILL language, see [Getting Started](#) in the *SKILL Language User Guide*.

## Using SKILL Code Examples

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

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

## Sample SKILL Code

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

### axlGetRunStatus

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

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

## Virtuoso Parameterized Cell SKILL Reference

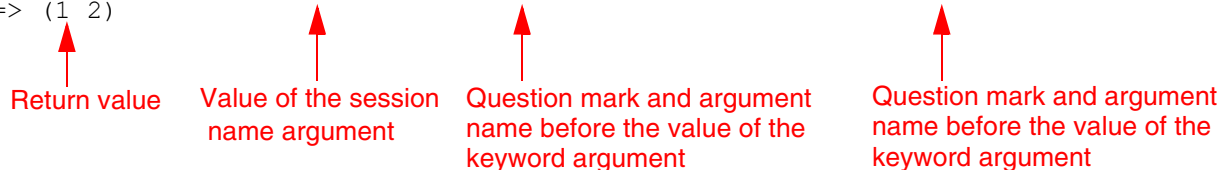
### Preface

---

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

#### Example

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



Return value

Value of the session name argument

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

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

## Accessing API Help

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

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

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

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

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



## Typographic and Syntax Conventions

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

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

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

## Identifiers Used to Denote Data Types

The Cadence SKILL language supports several data types to identify the type of value you can assign to an argument.

Data types are identified by a single letter followed by an underscore; for example, *t* is the data type in *t\_viewNames*. Data types and the underscore are used as identifiers only; they should not be typed.

---

Prefix	Internal Name	Data Type
<i>a</i>	array	array
<i>A</i>	amsobject	AMS Object
<i>b</i>	ddUserType	DDPI object
<i>B</i>	ddCatUserType	DDPI Category Object
<i>C</i>	opfcontext	OPF context
<i>d</i>	dbobject	Cadence database object (CDBA)
<i>e</i>	envobj	environment
<i>f</i>	flonum	floating-point number
<i>F</i>	opffile	OPF file ID
<i>g</i>	general	any data type
<i>G</i>	gdmSpecIIUserType	gdm spec
<i>h</i>	hdbobject	hierarchical database configuration object
<i>K</i>	mapiobject	MAPI object
<i>l</i>	list	linked list
<i>L</i>	tc	Technology file time stamp
<i>m</i>	nmpIIUserType	nmpII user type
<i>M</i>	cdsEvalObject	—
<i>n</i>	number	integer or floating-point number
<i>o</i>	userType	user-defined type (other)
<i>p</i>	port	I/O port
<i>q</i>	gdmSpecListIIIUserType	gdm spec list

## Virtuoso Parameterized Cell SKILL Reference

### Preface

---

Prefix	Internal Name	Data Type
<i>r</i>	defstruct	defstruct
<i>R</i>	rodObj	relative object design (ROD) object
<i>s</i>	symbol	symbol
<i>S</i>	stringSymbol	symbol or character string
<i>t</i>	string	character string (text)
<i>T</i>	txobject	Transient Object
<i>u</i>	function	function object, either the name of a function (symbol) or a lambda function body (list)
<i>U</i>	funobj	function object
<i>v</i>	hdbpath	—
<i>w</i>	wtype	window type
<i>x</i>	integer	integer number
<i>y</i>	binary	binary function
<i>&amp;</i>	pointer	pointer type

---

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

# **Virtuoso Parameterized Cell SKILL Reference**

## **Preface**

---

---

# Parameterized Cell Functions

---

This chapter provides information about creating parameterized cells (Pcells) using Cadence® SKILL language instead of using the graphical user interface. It includes safety rules for creating SKILL Pcells, information about Pcell master cells and submaster cells, and descriptions of supported `pc` functions.

Most of the supported `pc` SKILL functions listed in this chapter 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`

To read more about `pc` functions that you can use within Pcell code, see [Recommended, Supported SKILL Functions for Pcells](#) on page 24. For a description of the four `pc` Pcell functions listed above, see [pc Functions for SKILL Pcell Code](#) on page 31, later in this chapter.

**Note:** We recommend 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. For information about ROD, see the [\*Virtuoso Relative Object Design User Guide\*](#).

This section contains the following information:

[Safety Rules for Creating SKILL Pcells](#) [on page 22](#)

[Calling SKILL Procedures from within Pcells](#) [on page 23](#)

[Using Macros in Pcell SKILL Code](#) on page 23

[Physical Limits for Functions](#) on page 23

Recommended, Supported SKILL Functions for Pcells [on page 24](#)

About Pcell Super and Submaster Cells on page 29

pc Functions for SKILL Pcell Code on page 31

## Safety Rules for Creating SKILL Pcells

This section provides important rules for creating SKILL Pcells, including which functions are safe to use and what to avoid.



***If you use SKILL functions that are unsupported or not intended for use in Pcells, your Pcell code will be liable to fail when you try to translate your design to a format for use with a non-Virtuoso Cadence application or third-party application.***

The purpose of creating a Pcell is to *automate the creation of data*. Pcells should be designed as standalone entities, independent of the environment in which they are created and independent of the variety of environments in which you or someone else might want to use them. An environment can react to a Pcell, but Pcell code should not react to, interact with, or be dependent on an environment.

Although it is possible to create Pcells that depend on something in your current or local environment, and that use either unsupported or unrecommended functions, your Pcell code is likely to fail when you try to translate it for us in a different environment. And although it is possible to load, read, and write to files in the UNIX file system from within a Pcell, do not do so. You cannot control the file permission settings in other locations, so reading or writing from a Pcell can cause the Pcell to fail in other directories, other environments, and during evaluation by translators.

Functions that are not supported for use by customers within SKILL Pcells usually belong to specific applications (tools); they are unknown to other environments, to other tools, and to data translators. For example, if you create a Pcell in the `icfb` environment and include place-and-route functions, the Pcell will fail in the layout environment. Also, application-specific functions that are not supported for customer use can disappear or change without notice.

Create Pcells using only the recommended, supported functions. You can identify them by their prefixes. However, you can also use all of the basic SKILL language functions defined in the *Cadence SKILL Language Reference*; these functions do not have prefixes. To use only recommended, supported, documented functions, follow the rules in this section.

## Calling SKILL Procedures from within Pcells

You can call your own SKILL procedures from within Pcells. However, the code in procedures called by Pcells must follow all of the safety rules described in this section. The safety rules apply to all code that is evaluated when the system evaluates a Pcell, and that includes SKILL procedures called by the Pcell.

Called procedures are not compiled with the Pcell, so you must make sure they are loaded before the system evaluates the Pcell. You can do this by attaching the code files to your library; then the system automatically loads the procedures the first time it accesses the library.

For information about how to attach code files to a library, see “[How to Package a Pcell](#)” in the *Virtuoso Parameterized Cell Reference*.

## Using Macros in Pcell SKILL Code

Macros in Pcells are only expanded one level. You can include macros that do not reference or contain other macros (are not nested) in your Pcell SKILL code, and they will be expanded successfully.

To include a nested macro in a Pcell, it is recommended that you use an explicit function call in the Pcell code section of the `pcDefinePCell` function, and make sure that the function is defined and available (loaded) prior to any evaluation of the Pcell. This is usually done by including the function in your `libInit.il` file. If you do not use this method for nested macros, the nested code is treated as an undefined function, causing the Pcell evaluation to fail.

## Physical Limits for Functions

You can avoid creating a Pcell that is too large to be processed or displayed by not exceeding the physical limitations for SKILL functions. The following section is copied from the *Cadence SKILL Language User Guide, Chapter 3*, “Creating Functions in SKILL”:

The following physical limitations exist for functions:

- Total number of *required* arguments is less than 65536
- Total number of *keyword/optional* arguments is less than 255
- Total number of local variables in a `let` is less than 65536
- Total number of local variables in a `let` must be less than 65536

## Virtuoso Parameterized Cell SKILL Reference

### Parameterized Cell Functions

---

- Max size of code vector is less than 1GB

By default, code vectors are limited to functions that can compile less than 32KB words. This translates roughly into a limit of 20000 lines of SKILL code per function. The maximum number of arguments limit of 32KB is mostly applicable in the case when functions are defined to take an *@rest* argument or in the case of *apply* called on an argument list longer than 32KB elements.

### Recommended, Supported SKILL Functions for Pcells

When you create SKILL routines within Pcells, use only the following functions:

- The SKILL functions documented in the *Cadence SKILL Language Reference*; for example, *car*, *if*, *foreach*, *sprintf*, *while*.
- SKILL functions from the following families:

db	dd	cdf
rod	tech	abe
cst	tx	

- The following four *pc* SKILL functions, which are documented in this chapter:

[pcExprToString](#)

[pcFix](#)

[pcRound](#)

[pcTechFile](#)

You can use only these four *pc* SKILL functions because

- ❑ Most supported *pc* functions correspond to the graphical user interface commands. You can use them to create Pcells in the graphical Pcell environment, but you cannot use them in the body of SKILL Pcell code.
- ❑ Both the Pcell graphical user interface and the Pcell compiler are coded using *pc* SKILL functions. You cannot use the graphical user interface or compiler functions at all.

For a description of the four *pc* SKILL functions you can use in the body of Pcell code, see [“pc Functions for SKILL Pcell Code”](#) on page 31.



## Finding Supported SKILL Functions

You can use either the `listFunctions` command or the Cadence Finder quick reference tool to list supported functions. But remember, you cannot use most supported functions within your Pcell code.

**Note:** In releases prior to 4.4.3, the `listFunctions` command displayed a list of all SKILL functions. Starting with release 4.4.3, the `listFunctions` command displays only public SKILL functions. *Public SKILL functions* are supported by Cadence for use by customers. If you are using a software release prior to 4.4.3 and use the `listFunctions` command to display Pcell functions, the list returned is misleading because it includes `pc` functions used to code the graphical user interface and compiler.

You can use the Finder tool to quickly check which SKILL functions are documented and supported for the `db`, `dd`, `cdf`, `rod`, and `tech` SKILL families.

## Using the Finder Tool

To use the Finder tool, follow the steps below.

**1.** Start the Finder by doing one of the following:

- ☐ From within the Cadence Design Framework II product (DFII), choose *Tools – SKILL Development* and click *Finder*.
- ☐ In a UNIX window, type `cdsFinder`.

**2.** To display the supported SKILL functions,

- ☐ For *Searching*, select *All Available Finder Data*.
- ☐ For *Search String*, click *at beginning*.
- ☐ In the data entry area below *Search String*, type the function prefix. For example, type `tech`.
- ☐ Click *Search*.

All of the supported, documented SKILL functions for the prefix you entered appear in alphabetical order in the *Matches* field.

**3.** To see the arguments and a short description for a function that is listed in the *Matches* field, click the function name.

The syntax and a brief description of the selected function appear in the Descriptions window. You might need to scroll up to see the syntax.

4. If necessary, scroll up to see the syntax.

For functions that have keyword-value pairs for arguments, each argument probably wraps to the next line.

5. If the arguments are wrapping, make the Finder window wider by stretching the lower-left corner.
6. Before selecting another function, click the *Clear* button.

## Using print Functions in a Pcell

You cannot print or generate output from a Pcell, as it causes the Pcell evaluation to fail. However, you can bypass the system processing for print functions by using the `fprintf` or `println` function to print directly to `stdout` (standard output) in the format shown below:

```
fprintf( stdout ... )  
println( variable stdout )
```

where *variable* is your variable name. For example

```
fprintf( stdout "myVariable = %L \n" myVariable )
```

## Using printf or println in a Pcell

If you use the `printf` or `println` function in a Pcell, the output is treated as an error. In a cellview, errors are indicated by flashing markers. You can query the error in the cellview, change your Pcell code to fix the error, and recompile your Pcell.

To query markers, do the following:

1. Choose *Verify – Markers – Explain*.
2. Click a flashing marker.

A dialog box appears, showing the text printed from the Pcell by the `printf` or `println` function.

## Enclosing the Body of Code in a let or prog for Local Variables

When you use local variables in the body of SKILL Pcell code in a `pcDefinePCell` statement, be sure to enclose the Pcell code in a `let` or `prog` statement, and define all variables used in the Pcell code at the beginning of the `let` or `prog` statement. Defining variables as part of a `let` or `prog` prevents conflicts with variables used by the Pcell compiler.

Using `let` gives faster performance than `prog`; `prog` allows multiple exits while `let` exits only at its end.

## What to Avoid When Creating Pcells

When creating a Pcell, do not use functions with prefixes other than `db`, `dd`, `cdf`, `rod`, and `tech`. Although it is possible to call procedures with other prefixes from within a Pcell, you will not be able to view the Pcell in a different environment or translate it into the format required by another database. Most translators, including the physical design translators, can translate only basic SKILL functions; the four `pc` functions; and functions in the SKILL families `db`, `dd`, `cdf`, `rod`, and `tech`.

Specifically, **do not** use functions with application-specific prefixes such as `ael`, `abs`, `de`, `ge`, `hi`, `las`, `le`, `pr`, and `sch`. Procedures in any application-specific family are at a higher level of functionality than can be used in a Pcell safely. For example, if you create a Pcell using `le` functions, the Pcell works only in environments that include the Virtuoso layout editor. You cannot use a translator to export your design from the DFII database.

When a translator cannot evaluate a function, one of the following happens:

- The translator fails and issues an `undefined function` error message.
- The translator continues, issues a warning message, and translates the data incorrectly.

To make your design translate successfully, you must resolve undefined functions in Pcells. You can do this by flattening the Pcells, which results in a loss of hierarchy and parameterization, or by rewriting the Pcell code to eliminate the undefined functions.

### *Important*

Here are more important rules for creating Pcells. In your Pcell code,

- ☐ Do not prompt the user for input.
- ☐ Do not generate messages; message output is interpreted as an error.
- ☐ Do not load, read, or write to files in the UNIX file system.
- ☐ Do not run any external program that starts another process.
- ☐ If you need to drive external programs to calculate cell shapes, you can do this in SKILL. Use CDF callback procedures to save the resulting list of coordinate pairs in a string, and then pass the string as input to a SKILL Pcell.

This method has the advantage that the external program needs to be called only once per instance, not each time the design is opened. For more information about

## Virtuoso Parameterized Cell SKILL Reference

### Parameterized Cell Functions

---

callback procedures, refer to “[Using the Component Description Format](#)” in the *Virtuoso Parameterized Cell Reference*.

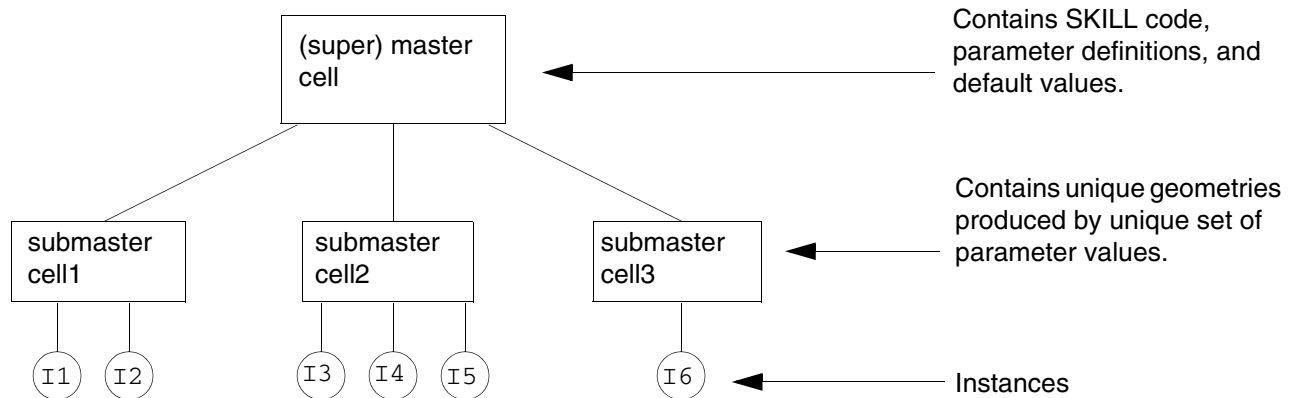
- ❑ Do not generate output with `print` functions, except as described in “[Using print Functions in a Pcell](#)” on page 26.

For more information about creating SKILL Pcells, also see

- The “[Creating SKILL Parameterized Cells](#)” chapter in the *Virtuoso Parameterized Cell Reference*
- How to use relative object design functions (with the prefix `rod`) to create geometry for Pcells in the *Virtuoso Relative Object Design User Guide*

## About Pcell Super and Submaster Cells

The master cell of a Pcell has two levels: a (super) master cell in the database and one or more temporary submaster cells in virtual memory. The Parameterized Cell software creates one submaster cell in virtual memory for each unique set of parameter values assigned to an instance of the (super) master cell.



A submaster cell resides in virtual memory for the duration of your editing session and is accessible by all cellviews. When you create an instance in a cellview, and a submaster with the same parameter values already exists in virtual memory, no new submaster is generated; the new instance references the existing submaster cell.

When you close all cellviews that reference a particular submaster cell, the database software automatically purges (removes) the referenced submaster cell from virtual memory but does not remove the master Pcell from your disk.

When you edit an editing session by exiting the Cadence software, the database software automatically purges (removes) all submaster cells from virtual memory. Purging does not remove master Pcells from your disk.

## Using the SKILL Operator ~> with Pcells

When you use the SKILL operator ~> to access information about the master cell of a Pcell instance, you must look two levels above the instance. If you look at only one level above, you access information about the Pcell submaster.

For example, for the instance I1 of the Pcell master mux2, you retrieve the database identity (dbId) of the mux2 Pcell master with the following statement:

```
I1~>master~>superMaster
```

## Virtuoso Parameterized Cell SKILL Reference

### Parameterized Cell Functions

---

But if you use the statement

```
I1~>master
```

you retrieve the *dbId* for the Pcell submaster.

## **pc Functions for SKILL Pcell Code**

When you create SKILL routines within Pcells, use only the following `pc` functions:

- `pcExprToString`
- `pcTechFile`
- `pcFix`
- `pcRound`

**Note:** See `pcRound` and `pcFix` under “Graphic Parameterized Cell SKILL Functions”.

## 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.

### Argument

<i>g_ilExpr</i>	SKILL expression you want to convert to a string.
-----------------	---

### Value Returned

<i>t_string</i>	The string equivalent of the SKILL expression, enclosed in quotes. If the value of the expression is <code>nil</code> or an error occurred, an empty string is returned.
-----------------	--

### Example

For the following expression, the system retrieves the value of the minimum width rule for the `metall` layer from the technology file and then converts that value to a string enclosed in quotes.

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



## 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.

### Argument

<i>g_expression</i>	The expression you want the system to evaluate when the Pcell containing the expression is evaluated.
---------------------	---

### Value Returned

<i>g_result</i>	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.
-----------------	---

### Example

In the following statement, the symbol `minWidth` is used only during evaluation of the expression; it is not treated as a Pcell parameter:

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

where `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.

# **Virtuoso Parameterized Cell SKILL Reference**

## Parameterized Cell Functions

---

---

## Graphical Parameterized Cell Functions

---

This chapter 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 SKILL Pcell code.

[Graphical Pcell Functions](#) on page 2

[Pcell Compiler Customization SKILL Functions](#) on page 148

[Parameterized Cell SKILL Cross-Reference Table](#) on page 164

## Graphical Pcell Functions

### 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

<i>l_pointList1</i>	First list of coordinates, specified as a list of two floating point numbers in the following format:  ' (x y)
<i>l_pointList2</i>	Second list of coordinates, specified as a list of two floating point numbers in the following format:  ' (x y)
<i>l_pointList3</i>	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.

#### Example

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

## **Virtuoso Parameterized Cell SKILL Reference**

### **Graphical Parameterized Cell Functions**

---

Verifies whether the three coordinates listed are collinear and orthogonal.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### pcConcatOrient

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

#### Description

Concatenates two specified object orientations into one.

#### Arguments

<i>t_orientation1</i>	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
<i>t_orientation2</i>	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

<i>t_concatOrient</i>	Orientation resulting from the concatenation.
"R0"	Returns "R0" if the concatenation fails.

#### Example

```
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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### Arguments

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

#### Value Returned

<i>d_condId</i>	Group ID used to store the details of the conditional inclusion.
<code>nil</code>	Returned if <i>d_cvId</i> does not identify a cellview or <i>l_figs</i> does not contain any objects.

#### Example

```
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`.



## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### 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

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

#### Value Returned

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

#### Example

```
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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### 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.

#### Prerequisites

You must have already defined parameters for the cellview.

#### Arguments

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

#### Value Returned

<i>t</i>	Parameterized (super) master was created.
<i>nil</i>	Parameterized (super) master was not created.

#### Example

```
pcDefineParamCell( cellviewId )
```

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

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### 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

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

#### Value Returned

<i>t</i>	Returned if the label is converted to a parameterized label.
<i>nil</i>	Returned if the label is not converted to a parameterized label.

#### Example

```
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 the example above the *dbId* 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

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

<i>d_paramLayerId</i>	Group ID used to record the details of the layer parameter.
-----------------------	---

### Example

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

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### Arguments

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

#### Value Returned

<i>d_paramShapeId</i>	Group ID used to record details of the parameterized paths.
-----------------------	---

#### Example

```
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

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

### Value Returned

<i>d_paramShapeId</i>	Group ID used to record details of the parameterized polygons.
-----------------------	--

### Example

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



## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

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

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

### Value Returned

<i>t</i>	Returned if the parameterized property is created.
<i>nil</i>	Returned if the parameterized property is not created.

### Example

```
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

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

### Value Returned

<i>d_paramShapeId</i>	Group ID used to record details of the parameterized rectangles.
-----------------------	--

### Example

```
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

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

### Value Returned

<i>d_refPointId</i>	Group ID used to record the details of the parameterized reference point.
---------------------	---

### Example

```
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.

## 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

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

### Value Returned

<i>d_refPointId</i>	Group ID used to store the details of the parameterized reference point.
---------------------	--

### Example

```
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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### 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 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`

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

*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).

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

- *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.

**Note:** Cadence encourages you to follow 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.

**Example 1:** ( Switch1 "float" 0.625 )

**Example 2:** ( Switch2 float 0.625 )

**Example 3:** ( Switch3 0.625 )



***For boolean arguments, you “must” specify the data type. It helps you to 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.



## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell 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. For more information about what to do and what to avoid when writing Pcell code, see [“Safety Rules for Creating SKILL Pcells”](#) on page 22.

### Value Returned

<code>d_cellViewId</code>	The cellview ID of the parameterized (super) master is returned if the (super) master is created.
<code>nil</code>	Returned if the parameterized (super) master cellview is not created.

### Example

```
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.

## 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

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

### Value Returned

<i>l_paramDefDPL</i>	The created parameter DPL.
----------------------	----------------------------

### Example

```
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`.

## **pcDefinePPCell**

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

### **Description**

Creates a parameterized Pcell master cellview (P<sup>2</sup>cell) 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<sup>2</sup>cell 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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### 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<sup>2</sup>cell definition.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

*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<sup>2</sup>cell definition.

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

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

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 `pcDefinePPCell` function.
- Call a user-defined function, which returns a Pcell parameter set. When P<sup>2</sup>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. For more information about what to do and what to avoid when writing Pcell code, see [“Safety Rules for Creating SKILL Pcells”](#) on page 22.

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

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### Value Returned

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

#### Example

```
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
      (
        (ddl "none" )
        (sdl "none" )
        (w 5.0)
        (l 1.0)
      )
    )
  )
)
```

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

```
        (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<sup>2</sup>cell method.



## **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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### Arguments

<i>d_cvId</i>	The database ID of the cellview in which the parameter applies.
<i>l_shapes</i>	List of the shapes replicated.
<i>l_namelist</i>	List of the symbols referenced in the step or repeat expressions. These become parameters of the Pcell.
<i>g_stepX</i>	SKILL expression specifying stepping distance in the horizontal direction.
<i>g_stepY</i>	SKILL expression specifying stepping distance in the vertical direction.
<i>g_repeatX</i>	SKILL expression specifying the number of times objects are repeated in the horizontal direction.
<i>g_repeatY</i>	SKILL expression specifying the number of times objects are repeated in the vertical direction.
<i>g_stretchX</i>	Name of the vertical, dependent stretch control line associated with this repetition.
<i>g_stretchY</i>	Name of the horizontal, dependent stretch control line associated with this repetition.
<i>g_adjustX</i>	SKILL expression specifying the adjustment from the reference dimension to be applied to the vertical, dependent stretch control line.
<i>g_adjustY</i>	SKILL expression specifying adjustment from the reference dimension to be applied to the horizontal, dependent stretch control line.
<i>t_direction</i>	Direction of repetition.  Valid Values: horizontal, vertical, horizontal and vertical

#### Value Returned

<i>d_repeatId</i>	Group ID used to store details of the repetition.
-------------------	---

#### Example

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

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

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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### Arguments

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

#### Value Returned

<i>d_stepObjectId</i>	Group ID used to record the details of the repetition along the parameterized shape.
-----------------------	--

#### Example

```
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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### Arguments

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

#### Value Returned

<i>d_StretchId</i>	Group ID used to store the details of the stretch parameter.
--------------------	--

#### Example

```
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**

<i>d_groupId</i>	Group ID of the conditionally included objects.
------------------	---

### **Value Returned**

<i>t</i>	Returned if the parameter is deleted.
<i>nil</i>	Returned if <i>d_groupId</i> is not a conditional inclusion parameter.

### **Example**

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

Deletes all conditional inclusion parameters in the cellview *cv*.



## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### 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](#) on page 120 instead of this function.

#### Prerequisites

You must have defined the named parameter.

#### Arguments

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

#### Value Returned

<i>t</i>	Returned if the parameter definition is deleted.
<i>nil</i>	Returned if <i>t_paramName</i> is not a valid parameter name.

#### Example

```
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.

### **Prerequisites**

You must have defined the layer parameter.

### **Arguments**

<i>d_groupId</i>	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 <i>d_groupId</i> is not a valid identifier for a layer parameter.

### **Example**

```
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

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

### Value Returned

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

### Example

```
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.

### Prerequisites

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

### Arguments

<i>d_memberId</i>	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.

### Example

```
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

<i>d_groupId</i>	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 <i>d_groupId</i> is not a valid identifier for a parameterized reference point.

### Example

```
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

<i>d_groupId</i>	The database ID of the repetition parameter you want to delete.
------------------	---

### Value Returned

t	Returned if the parameter is deleted.
nil	Returned if <i>d_groupId</i> is not a repeat identifier.

### Example

```
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**

<i>d_groupId</i>	The database ID of the repetition along a shape as returned by <code>pcDefineSteppedObject</code> .
------------------	---

### **Value Returned**

<i>t</i>	Returned if the parameter is deleted.
<i>nil</i>	Returned if <i>d_groupId</i> is not a repetition-along-shape identifier.

### **Example**

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

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

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

## 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

<i>g_device</i>	A SKILL++ Pcell class object that is inherited from class <code>pcParamClass</code> .
-----------------	---

## Value Returned

<i>t</i>	If the database objects are created for a specified SKILL++ Pcell class.
<i>nil</i>	If the database objects are not created.

## Example

```
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
```



## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

```
        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

<i>txfl_argument</i>	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

<i>l_typeValue</i>	List containing the following two elements: data type and value, where the data type can be a string, integer, or floating-point number.
--------------------	--

<i>nil</i>	Returned if <i>txfl_argument</i> has an invalid data type or contains an invalid expression.
------------	--

### Example 1

```
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.

### Example 2

```
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

<i>l_pointList</i>	List of coordinates in one of the following formats:  <code>list( '( x1 y1 ) '( x2 y2 ) ... '( xn yn ) )</code> <code>'( x1:y1 x2:y2 ... xn:yn )</code>
--------------------	--

### Value Returned

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

### Example

```
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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### 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.

For example, 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 <i>x</i> equals	pcFix( <i>x</i> ) 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 <i>x</i> equals	And <i>f_precision</i> equals	Then pcFix returns
14.998	0.001	14

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

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

### Arguments

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

### Value Returned

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

### Example

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
```

## **Virtuoso Parameterized Cell SKILL Reference**

### **Graphical Parameterized Cell Functions**

---

```
    ; perform conditional operations here  
) ;endif
```

## 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

<i>l_pointList</i>	List of three coordinates in one of the following formats:  <pre>list( '( x1 y1 ) '( x2 y2 ) '( x3 y3 ) ) '( x1:y1 x2:y2 x3:y3 )</pre>
--------------------	--

### Value Returned

<i>x_signedDegrees</i>	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.
<i>nil</i>	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.

### Example

```
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

<i>d_cellViewId</i>	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

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

### Example

```
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

<i>d_cellViewId</i>	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.
<i>S_codeParamName</i>	The name of any defined code parameter. Type: Symbol or String.

### Value Returned

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

### Example

```
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.

### Prerequisites

You must have defined the conditional inclusion parameters.

### Arguments

<i>d_cvId</i>	The database ID of the cellview containing conditional inclusion parameters.
---------------	--

### Value Returned

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

### Example

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

Lists all conditional inclusion parameters defined in *cv*.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### pcGetDefaultParamsFromClass

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

#### Description

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

#### Arguments

<i>s_className</i>	A class symbol declared for a class name of a Pcell.
--------------------	--

#### Value Returned

<i>l_paramList</i>	The parameter list that contains ( <i>paramName paramType value</i> ).
--------------------	--

#### Example

```
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.

### Prerequisites

You must have defined the inherited parameter.

### Arguments

<i>d_instId</i>	The database ID of the instance of the Pcell whose inherited parameter is to be retrieved.
-----------------	--

### Value Returned

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

### Example

```
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.

### Prerequisites

You must have defined the inherited parameter.

### Arguments

<i>d_cvId</i>	The database ID of the cellview whose inherited parameters you want listed.
---------------	---

### Value Returned

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

### Example

```
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

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

### Value Returned

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

### Example

```
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))
```

## **Virtuoso Parameterized Cell SKILL Reference**

### **Graphical Parameterized Cell Functions**

---

(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

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

### Value Returned

<i>l_vertex_points</i>	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: <code>(( x y) (x y) ... (x y))</code>
<i>nil</i>	The polygon was not found or the offset was not applied successfully.

### Example

```
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 )
```



## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

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)
)
```

## 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

<i>g_device</i>	A SKILL++ Pcell class object that is inherited from class <code>pcParamClass</code> .
<i>s_propName</i>	A parameter slot name of the given device.

### Value Returned

<i>t_paramType</i>	The type contained in the parameter slot <code>propName</code> of the given device.
--------------------	---

### Example

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

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

## **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*.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### Arguments

<i>d_cvId</i>	The database ID of the cellview containing the parameterized path.
<i>l_objectVertices</i>	List of coordinates determining the object that is offset from the parameterized path, in one of the following formats: <code>list( '( x1 y1 ) '( x2 y2 ) ... '( xn yn ) )</code> <code>list( x1:y1 x2:y2 ... xn:yn )</code>
<i>l_pathVertices</i>	List of coordinates identifying the parameterized path, in one of the following formats: <code>list( '( x1 y1 ) '( x2 y2 ) ... '( xn yn ) )</code> <code>list( x1:y1 x2:y2 ... xn:yn )</code>
<i>t_endpoint</i>	String specifying an endpoint of the parameterized path.  Valid Values: "first" or "last"  Default: none
<i>l_offset</i>	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 <code>nil</code>  Default: none

#### Value Returned

<i>l_newObjVertices</i>	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:  <code>( (x y) (x y) ... (x y) )</code>  where x and y can be an integer or floating-point number.
<code>nil</code>	The function did not complete successfully.

#### Example

```
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 )
```

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

```
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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### pcGetParameters

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

#### Description

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

**Note:** You must have defined the parameters in this cellview.

#### Arguments

<i>d_cvId</i>	The database ID of the cellview containing the parameters.
---------------	--

#### Value Returned

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

#### Example

```
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.

### Prerequisites

You must have defined a parameterized label.

### Arguments

<i>d_labelId</i>	The database ID of the label whose parameterized label is to be retrieved.
------------------	--

### Value Returned

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

### Example

```
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.

### Prerequisites

You must have defined a parameterized label.

### Arguments

<i>d_cvId</i>	The database ID of the cellview containing parameterized labels.
---------------	--

### Value Returned

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

### Example

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

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



## pcGetParamLayers

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

### Description

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

### Prerequisites

You must have defined layer parameters for this cellview.

### Arguments

<i>d_cvId</i>	The database ID of the cellview whose layer parameters you want listed.
---------------	---

### Value Returned

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

### Example

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

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

## pcGetParamLayerDefn

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

### Description

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

### Prerequisites

You must have defined a layer parameter for this shape.

### Arguments

<i>d_instId</i>	The database ID of the shape whose layer parameter is to be retrieved.
-----------------	--

### Value Returned

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

### Example

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

Retrieves the layer parameters defined in *shape*.

## pcGetParamProps

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

### Description

Lists parameterized properties in the specified cellview.

### Prerequisites

You must have defined the parameterized properties.

### Arguments

<i>d_cvId</i>	The database ID of the cellview containing parameterized properties.
---------------	--

### Value Returned

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

### Example

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

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

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### 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`.

**Note:** You must have defined the parameterized shape.

#### Arguments

<i>d_instId</i>	The database ID of the shape whose parameterized shape parameter is to be retrieved.
-----------------	--

#### Value Returned

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

#### Example

```
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.

### Prerequisites

You must have defined a parameterized shape for this cellview.

### Arguments

<i>d_cvId</i>	The database ID of the cellview containing the shape parameters you want listed.
---------------	--

### Value Returned

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

### Example

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

Lists all shape parameters defined in *cv*.

## pcGetParamSlotValue

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

### Description

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

### Arguments

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

### Value Returned

<i>g_value</i>	The value of the parameter slot, <i>propName</i> , of the given device.
----------------	---

### Example

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

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

## pcGetRefPointDefn

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

### Description

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

### Prerequisites

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

### Arguments

<i>d_objectId</i>	The database ID of the object with the reference point parameter.
-------------------	---

### Value Returned

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

### Example

```
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.

### Prerequisites

You must have defined parameters in this cellview.

### Arguments

<i>d_cvId</i>	The database ID of the cellview whose reference point parameters you want listed.
---------------	---

### Value Returned

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

### Example

```
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.

### Prerequisites

You must have defined repetition parameters for the cellview.

### Arguments

<i>d_objectId</i>	The database ID of the object with repetition parameters.
-------------------	---

### Value Returned

<i>l_repeatlist</i>	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.
<i>nil</i>	Returned if <i>d_objectId</i> does not identify an object or there are no repetition parameters defined with it.

### Example

```
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.

### Prerequisites

You must have defined repetition parameters for the cellview.

### Arguments

<i>d_cvId</i>	The database ID of the cellview with repetition parameters.
---------------	---

### Value Returned

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

### Example

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

Lists all repetition parameters defined in *cv*.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### 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

<i>l_pointList</i>	List of coordinates in either of the following formats: <code>'( ( x1 y1 ) ( x2 y2 ) ) ( x1:y1 x2:y2 )</code>
--------------------	--

#### Value Returned

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

#### Example

```
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.

### Prerequisites

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

### Arguments

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

### Value Returned

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

### Example

```
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.

### Prerequisites

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

### Arguments

<i>d_cvId</i>	The database ID of the cellview containing the repetition-along-shape parameters.
---------------	---

### Value Returned

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

### Example

```
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.

**Note:** You must have defined this line as a stretch control line.

### Arguments

<i>d_objectId</i>	The database ID of the line with the stretch parameter.
-------------------	---

### Value Returned

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

### Example

```
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.

### Prerequisites

You must have defined the stretch control lines.

### Arguments

<i>d_cvId</i>	The database ID of the cellview containing the stretch parameters.
---------------	--

### Value Returned

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

### Example

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

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

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### 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

<i>d_cvId</i>	The database ID of the cellview containing a Pcell master with stretch lines.
---------------	---

#### Value Returned

<i>l_stretchLines</i>	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.
<i>nil</i>	The function did not execute successfully.

#### Example

```
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

<i>l_pointList</i>	List of coordinates defining the original rectangular box, in either of the following formats:  <pre>'( ( x1 y1 ) ( x2 y2 ) ) ( x1:y1 x2:y2 )</pre>
<i>xf_margin</i>	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

<i>l_incrementedPointList</i>	List of coordinates defining the incremented box, in either of the following formats:  <pre>'( ( x1 y1 ) ( x2 y2 ) ) ( x1:y1 x2:y2 )</pre>
<i>nil</i>	The function did not complete successfully.

### Example

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

## **Virtuoso Parameterized Cell SKILL Reference**

### **Graphical Parameterized Cell Functions**

---

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

<i>l_pointList</i>	List of coordinates defining the original manhattan polygon, in either of the following formats:  <pre>'( ( x1 y1 ) ( x2 y2 ) ... ( xn yn ) ) ( x1:y1 x2:y2 ... xn:xy )</pre>
<i>xf_margin</i>	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

<i>l_incrementedPointList</i>	List of coordinates defining the incremented manhattan polygon, in either of the following formats:  <pre>'( ( x1 y1 ) ( x2 y2 ) ... ( xn yn ) ) ( x1:y1 x2:y2 ... xn:xy )</pre>
<i>nil</i>	The function did not complete successfully.

### Example

```
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)
```

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

```
(-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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### 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.

#### Prerequisites

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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### 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.

### Prerequisites

You must have already defined parameters for the cellview.

### Arguments

<i>l_cellIdentifier</i>	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.

### Example

```
pcHIDefineParamCell( cellview_Id )
```

Displays the Compile To Pcell form.

## **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.

### **Prerequisites**

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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### 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

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

### Value Returned

<i>d_repeatId</i>	Group ID used to store details of the repetition.
-------------------	---

### Example

```
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

<i>t_direction</i>	Direction of the stretch.  Valid Values: right, left, rightAndLeft, up, down, upAndDown
--------------------	---

### Value Returned

<i>d_StretchId</i>	Group ID used to store the details of the stretch parameter.
--------------------	--

### Example

```
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.

## **pcHDeleteLayer**

```
pcHDeleteLayer()  
=> 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.

## **pcHDeleteParameterizedShape**

```
pcHDeleteParameterizedShape()  
=> 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.

## **pcHDeleteRefPointObject**

```
pcHDeleteRefPointObject()  
=> 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.

## **pcHDeleteSteppedObject**

```
pcHDeleteSteppedObject()  
=> 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

<code>t</code>	Highlights inclusion groups and displays the Show Conditional Inclusion text window.
<code>nil</code>	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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### 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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### 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**

<code>t</code>	The <i>Show Parameters</i> text window displays or there are no parameters defined for the Pcell.
<code>nil</code>	There are no parameters defined for the Pcell or if the <i>Show Parameters</i> 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**

<code>t</code>	The Show Reference Point text window displays or there are no reference point groups.
<code>nil</code>	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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### **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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### 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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### **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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### **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**

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

### **Value Returned**

<code>t</code>	If the specified <code>propName</code> is a parameter slot of the specified device.
<code>nil</code>	If the specified <code>propName</code> is not a parameter slot of the specified device.

### **Example**

```
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

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

### Value Returned

<i>d_paramId</i>	Group ID of the property that stores the parameter.
<i>nil</i>	Returned if the function does not execute.

### Example

```
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.

**Note:** You must have defined the stretch control line.

### Arguments

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

### Value Returned

<i>d_StretchId</i>	Group ID used to store stretch parameter details.
<code>nil</code>	Returned if <i>d_lineId</i> is not defined as a stretch control line.



## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### Example

```
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

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

### Value Returned

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

### Example

```
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

<i>n_num</i>	Any number you want to round.
<i>f_precision</i>	Floating-point number specifying number of decimal places to the right of the decimal point to use as a range for determining whether <i>n_num</i> is close to a whole number.  Default: 0.001
<i>x_tolerance</i>	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

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

### Example

The examples below show numbers rounded using `pcRound`.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

```
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
```

## pcSetFTermWidth

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

### Description

Creates a net or terminal name and assigns a width.

### Arguments

<i>t_baseName</i>	String specifying the base name for the net or terminal.
<i>x_width</i>	Integer specifying the width of the net or terminal.

### Value Returned

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

### Example

```
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**

<i>g_device</i>	An object of the SKILL++ Pcell class, which is inherited from the class <code>pcParamClass</code> .
<i>d_cv</i>	A Pcell super master ID or Pcell sub master ID.

### **Value Returned**

<i>t</i>	If the class slot values of the specified device are set.
<i>nil</i>	Unable to set the device's class slot values.

### **Example**

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

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

## 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

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

### Value Returned

<i>g_value</i>	The value of the parameter slot, <code>propName</code> , of a given device.
----------------	---

### Example

```
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.

### Prerequisites

You must have defined the parameters in this cellview.

### Arguments

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

### Value Returned

<i>t</i>	The SKILL file was created.
<i>nil</i>	The SKILL file was not created.

### Example

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



## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### Arguments

<i>d_shape1</i>	The database ID of the shape to be replicated.
<i>l_stepDetails</i>	<p>Disembodied property list, which is a list that starts with a SKILL data object, in this case, <code>nil</code>, followed by alternating symbol name/value pairs, and is not attached to a specific object or symbol. The attributes contained in the list are:</p> <pre>nil 'step          xf_stepValue 'gap           xf_gapValue 'startOffset   xf_startValue 'endOffset     xf_endValue 'objType       t_objTypeValue l_shape2Points</pre> <p>List of coordinates specifying the second shape, in one of the following formats:</p> <pre>list( '( x1 y1 ) '( x2 y2 ) ... '( xn yn ) ) '( x1:y1 x2:y2 ... xn:yn )</pre>

#### Value Returned

<code>t</code>	The shape, <i>d_shape1Id</i> , was replicated successfully.
<code>nil</code>	The shape, <i>d_shape1Id</i> , was not replicated successfully.

#### Example

```
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.

## 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 the Make Ultra Pcell Command in the *Virtuoso Parameterized Cell Reference*.

### Prerequisites

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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

#### Arguments

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

#### Value Returned

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

#### Example

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

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

## **Pcell Compiler Customization SKILL Functions**

This section provides syntax, descriptions, and examples for the SKILL functions associated with customizing the Pcell compiler. Refer to “Customizing the Pcell Compiler” in the *Parameterized Cell Reference Guide* for more information.

## **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**

<i>p_port</i>	Port to which the output code is generated.
---------------	---

### **Value Returned**

t   nil	Return value is not relevant.
---------	-------------------------------

### **Example**

```
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

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

### Value Returned

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

### Example

```
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

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

### Value Returned

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

### Example

```
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

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

### Value Returned

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

### Example

```
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
```

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

---

```
; 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

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

### Value Returned

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

### Example

```
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

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

### Value Returned

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

### Example

```
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

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

### Value Returned

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

### Example

```
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

<i>d_shape</i>	Database ID of shape.
<i>p_port</i>	Port to which the output code is generated.

### Value Returned

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

### Example

```
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

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

### Value Returned

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

### Example

```
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

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

### Value Returned

t   nil	Return value is not relevant.
---------	-------------------------------

### Example

```
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**

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

### **Value Returned**

<i>t</i>   <i>nil</i>	Return value is not relevant.
-----------------------	-------------------------------

### **Example**

```
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

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

### Value Returned

<i>t</i>   <i>nil</i>	Return value is not relevant.
-----------------------	-------------------------------

### Example

```
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**

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

### **Value Returned**

<i>t / nil</i>	Return value is not relevant.
----------------	-------------------------------

### **Example**

```
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

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

### Value Returned

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

### Example

```
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.

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

## 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 SKILL Function	Procedural SKILL Function	Menu Command
<u><code>pcHIDefineParamCell</code></u>	<u><code>pcDefinePCell</code></u>	<i>Compile – To Pcell</i>
<u><code>pcHICompileToSkill</code></u>	<u><code>pcSkillGen</code></u>	<i>Compile – To Skill File</i>
<u><code>pcHIDefineCondition</code></u>	<u><code>pcDefineCondition</code></u> <u><code>pcGetConditions</code></u> <u><code>pcGetParameters</code></u>	<i>Conditional Inclusion – Define</i>
<u><code>pcHIDeleteCondition</code></u>	<u><code>pcDeleteCondition</code></u> <u><code>pcDefineCondition</code></u> <u><code>pcGetConditions</code></u>	<i>Conditional Inclusion – Delete</i>
<u><code>pcHIModifyCondition</code></u>	No procedural function	<i>Conditional Inclusion – Modify</i>
<u><code>pcHIDisplayCondition</code></u>	No procedural function	<i>Conditional Inclusion – Show</i>
<u><code>pcHIDefineInheritedParameter</code></u>	<u><code>pcDefineInheritParam</code></u> <u><code>pcGetInheritParamDefn</code></u> <u><code>pcGetParameters</code></u>	<i>Inherited Parameters – Define/Modify</i>
<u><code>pcHIDisplayInheritedParameter</code></u>	No procedural SKILL function	<i>Inherited Parameters – Show</i>
<u><code>pcHIDefineLabel</code></u>	<u><code>pcDefineParamLabel</code></u> <u><code>pcGetParameters</code></u> <u><code>pcGetParamLabelDefn</code></u> <u><code>pcGetParamLabels</code></u>	<i>Parameterized Label – Define</i>
<u><code>pcHIModifyLabel</code></u>	No procedural function	<i>Parameterized Label – Modify</i>

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

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

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

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



## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

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

## Virtuoso Parameterized Cell SKILL Reference

### Graphical Parameterized Cell Functions

Interactive SKILL Function	Procedural SKILL Function	Menu Command
—	—	<i>Connectivity – Nets – Add Shape</i>
—	—	<i>Connectivity – Nets – Remove Shape</i>
<u>auHiUltraPCell</u>	No procedural function	<i>Make Ultra Pcell</i>

---

## Express Pcells Data Management Functions

---

This chapter describes SKILL functions for Express Pcells data management. These functions require the VLS L license in IC6.1.8 and the VLS XL license in ICADVM20.1 (for both reading and writing the cache). Additionally, these SKILL functions operate only when the Express Pcell feature is enabled by setting the `CDS_ENABLE_EXP_PCELL` environment variable to `true`.

## **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**

<i>t_libName</i>	Specifies the name of the library
<i>t_cellName</i>	Specifies the name of the cell
<i>t_viewName</i>	Specifies the name of the view

## Virtuoso Parameterized Cell SKILL Reference

### Express Pcells Data Management Functions

---

#### 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.

## 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.

### Arguments

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

## 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.

### Arguments

<i>t_libName</i>	Name of the library.
<i>t_cellName</i>	Name of the cell.
<i>t_viewName</i>	Name of the view.
<i>n_depth</i>	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

<i>t</i>	Successfully saved all the submasters to the cache.
<i>nil</i>	Unable to save the submasters to the cache.

### Example

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

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

## 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.

**Note:** 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.***

### Arguments

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



## **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.

## xpcDumpCache

```
xpcDumpCache (
    g_detailReport
    [ filename ]
)
```

### Description

This SKILL API dumps the Express Pcell cache information on CIW and the file, which is specified as an argument.

**Note:** The value that you specify for this argument should be enclosed in quotation marks.

### Arguments

<i>g_detailReport</i>	When set to t, a <u>Detailed Report</u> is generated. When set to nil, the <u>Short Report</u> is generated.
<i>filename</i>	This is an optional argument. If the file name is specified and is in the write mode then the cache information is dumped on both CIW and the file specified. Otherwise, the cache information is dumped only on CIW.

**Note:** Ensure that environment variables, CDS\_ENABLE\_EXP\_PCELL and CDS\_EXP\_PCELL\_DIR are set appropriately.

### Short Report

It lists all supermasters represented in cache, followed by the number of submasters (stored in the cache) for each supermaster as shown in the following examples:

#### Example 1

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

In this example the cache information is dumped in both CIW and the file1.dump file as shown below.

```
Express PCell Cache Dump Report:
```

```
-----
SuperMaster(lib/cell/view)      Number of submasters
```

## Virtuoso Parameterized Cell SKILL Reference

### Express Pcells Data Management Functions

---

PDKLIB/p/layout	1
PDKLIB/n/layout	1

#### Example 2

```
xpcDumpCache( nil )
```

In this example the cache information is dumped only in CIW as shown below.

```
Express PCell Cache Dump Report:
```

```
-----  
SuperMaster(lib/cell/view)          Number of submasters  
testLib/customCell/layout          5
```

#### Detailed Report

In addition to the information provided in the short report, it lists each submaster in the cache with the list of parameter names and values as shown in the example below:

#### Example

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

In this example, the cache information is dumped in both CIW and the `file2.dump` file as shown below.

```
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
```

## Virtuoso Parameterized Cell SKILL Reference

### Express Pcells Data Management Functions

---

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