Virtuoso Relative Object Design SKILL Reference

Product Version ICADVM20.1
October 2020

Contents

Preface

Virtuoso® relative object design (ROD) is a set of high-level functions for defining simple to complex layout objects and their relationships to each other, without the need to use low-level Cadence® SKILL language functions.

This user guide is aimed at CAD engineers and assumes that you are familiar with the development and design of integrated circuits, Virtuoso® Layout Suite L editor, and SKILL programming. This user guide also assumes that you are familiar with the creation of parametrized cells using SKILL or the graphical user interface.

Scope

Unless otherwise noted, the functionality described in this guide can be used in both mature node (for example, IC6.18) and advanced node (for example, ICADVM20.1) releases.

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

Licensing Requirements

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

Related Documentation

What's New and KPNS

- *Virtuoso Relative Object Design What's New*
- *Virtuoso Relative Object Design Known Problems and Solutions*

Installation, Environment, and Infrastructure

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

Technology Information

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

Virtuoso Tools

IC6.1.8 Only

- *Virtuoso Layout Suite L User Guide*
- *Virtuoso Layout Suite XL User Guide*
- *Virtuoso Layout Suite GXL Reference*

ICADVM20.1 Only

- *Virtuoso Layout Viewer User Guide*
- *Virtuoso Layout Suite XL: Basic Editing User Guide*
- *Virtuoso Layout Suite XL: Connectivity Driven Editing Guide*
- *Virtuoso Layout Suite EXL Reference*
- *Virtuoso Concurrent Layout User Guide*
- *Virtuoso Design Planner User Guide*
- *Virtuoso Electromagnetic Solver Assistant User Guide*
- *Virtuoso Multi-Patterning Technology User Guide*
- *Virtuoso Placer User Guide*
- *Virtuoso RF Flow Guide*
- *Virtuoso Simulation Driven Interactive Routing User Guide*
- *Virtuoso Width Spacing Patterns User Guide*

IC6.1.8 and ICADVM20.1

- *Virtuoso Abstract Generator User Guide*
- *Virtuoso Custom Digital Placer User Guide*
- *Virtuoso Design Rule Driven Editing User Guide*
- *Virtuoso Electrically Aware Design Flow Guide*
- *Virtuoso Floorplanner User Guide*
- *Virtuoso Fluid Guard Ring User Guide*
- *Virtuoso Interactive and Assisted Routing User Guide*
- *Virtuoso Layout Suite SKILL Reference*
- *Virtuoso Module Generator User Guide*
- *Virtuoso Parameterized Cell Reference*
- *Virtuoso Pegasus Interactive User Guide*
- *Virtuoso Space-based Router User Guide*

- *Virtuoso Symbolic Placement of Devices User Guide*
- *Virtuoso Voltage Dependent Rules Flow Guide*

Relative Object Design and Inherited Connections

- *Virtuoso Relative Object Design User Guide*
- *Virtuoso Schematic Editor User Guide*

Additional Learning Resources

Video Library

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

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

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

Virtuoso Videos Book

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

Rapid Adoption Kits

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

To explore the full range of training courses provided by Cadence in your region, visit Cadence Training or write to training_enroll@cadence.com.

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

Help and Support Facilities

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

- The Virtuoso *Help* menu provides consistent help system access across Virtuoso tools and applications. The standard Virtuoso *Help* menu lets you access the most useful help and support resources from the Cadence support and corporate websites directly from the CIW or any Virtuoso application.
- The Virtuoso Welcome Page is a self-help launch pad offering access to a host of useful knowledge resources, including quick links to content available within the Virtuoso installation as well as to other popular online content.

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

For more information, see Getting Help in *Virtuoso Design Environment User Guide.*

Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

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

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

Feedback about Documentation

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

- Find erroneous information in a product manual
- Cannot find in a product manual the information you are looking for

- Face an issue while accessing documentation by using Cadence Help

You can also submit feedback by using the following methods:

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

On the Cadence Online Support Product Manuals page, select the required product and submit your feedback by using the *Provide Feedback* box.The following documents contain information about related tools and the SKILL language.

Understanding Cadence SKILL

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

For more information about the SKILL language, see Getting Started in the *SKILL Language User Guide*.

Using SKILL Code Examples

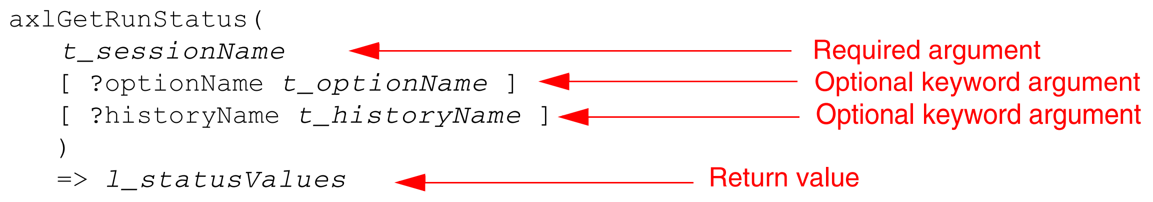
The SKILL APIs in this user manual are explained with illustrative code examples.

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

Sample SKILL Code

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

axlGetRunStatus

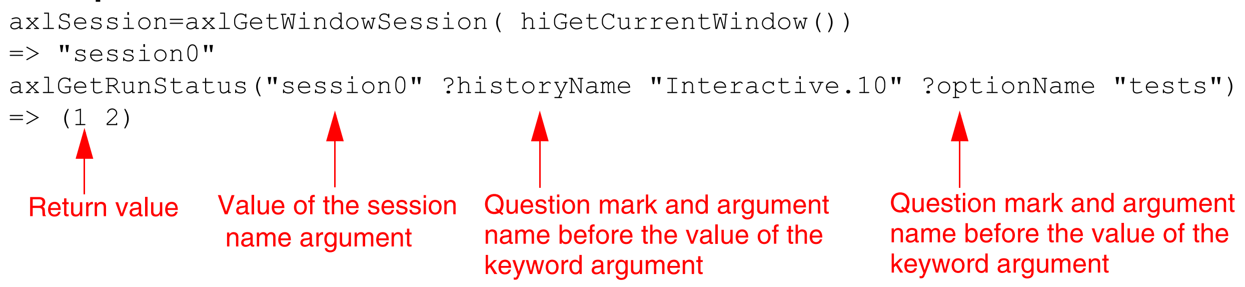


```
axlGetRunStatus(
    t_sessionName
    [ ?optionName t_optionName ]
    [ ?historyName t_historyName ]
    )
    => l_statusValues
```

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

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

Example



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

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

Accessing API Help

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

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

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

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

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

Typographic and Syntax Conventions

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

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

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

Identifiers Used to Denote Data Types

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

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

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

1

Relative Object Design Functions

This chapter contains information you need to know before using the Virtuoso® relative object design (ROD) functions and describes the ROD functions.

For more information on relative object design concepts, refer to the *Virtuoso Relative Object Design User Guide*.

Important Information related to Using ROD SKILL Functions

Editing ROD Objects with the Virtuoso Layout Editor

Using Variables in ROD Functions

Maximum String Length for ROD Function Arguments

List of ROD SKILL Functions

Functions are listed in alphabetical order

rodAddMPPChopHole

rodAddPoints

rodAddSubPart

rodAddToX

rodAddToY

rodAlign

rodAssignHandleToParameter

rodCheck

rodUnNameShape

Editing ROD Objects with the Virtuoso Layout Editor

For information about how the Virtuoso layout editor commands work with Virtuoso relative object design (ROD) objects, see Appendix E, "How Virtuoso Layout Editor Works with ROD Objects" in the *Virtuoso Relative Object Design User Guide*.

Using commands that are not fully supported for ROD objects could cause the objects to lose the ROD information associated with them, changing the objects into ordinary shapes.

Using Variables in ROD Functions

The following variables are useful in ROD functions:

- `pcCellView`

  An internal variable automatically created by the `pcDefinePCell` function. `pcCellView` contains the database ID of the cellview you are creating. Within the body of your Pcell code, use the `pcCellView` variable as the cellview identifier for which you create objects. In SKILL expressions (`ILExpr`) for ROD arguments, use the `rodCellView` variable rather than `pcCellView`.

- `tcCellView`

  An internal variable automatically created by the `tcCreateCDSDeviceClass` function. `tcCellView` contains the database ID of the cellview you are creating. Within the body of your Pcell code, use the `tcCellView` variable as the cellview identifier for which you create objects. In SKILL expressions (`ILExpr`) for ROD arguments, use the `rodCellView` variable rather than `pcCellView`.

- `rodCellView`

  When used in a SKILL expression (`ILExpr`) for a ROD argument, such as when specifying the *`n_xSep`* and *`n_ySep`* arguments for `rodCreatePath`, `rodCellView` contains the database ID of the current cellview. Using `rodCellView` allows you to reference rules in your technology file so that your code will work in different technologies. For example, you can create alignments at the zero-level in your hierarchy, and the system automatically updates the alignments when you make changes to technology file rules that affect them. You can use `rodCellView` inside or outside of Pcells.

Maximum String Length for ROD Function Arguments

The maximum number of characters you can specify for a ROD function argument that accepts a string is 1,024. If you specify a string having more than 1,024 characters, you get a syntax error. When you assign a string value to any of the following name arguments, use less than 1,024 characters:

- Paths, rectangles, or polygons when you create them
- Objects using the `rodNameShape` function
- User-defined handles

If you specify a name using a symbol, use no more than 255 characters in the symbol name. Any symbol name longer than 255 characters results in an error.

Using Special Characters in Names

When possible, avoid using special characters in the names of handles or ROD objects. Using a special character requires you to type a backslash as an escape character every time you type the name.

If you must use a special character in a string, type a backslash followed by the code for the special character, as shown in the following table:

Table 1-1 Using Special Characters in ROD Object Names

Special Character	Escape Sequence
backspace	`\b`
form feed	`\f`
new line (line feed)	`\n`
carriage return	`\r`
horizontal tab	`\t`
vertical tab	`\v`
backslash	`\\`
single quotation mark	`\'`
double quotation mark	`\"`

rodAddMPPChopHole

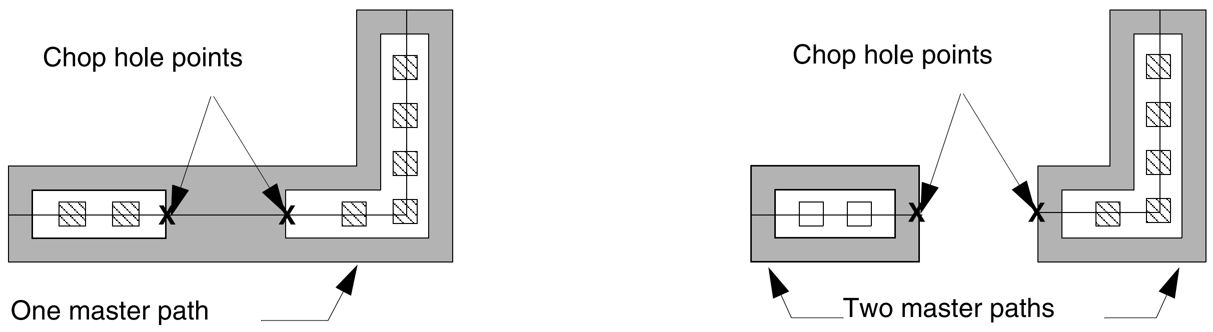
```
rodAddMPPChopHole(
     R_rodObj
     l_pts
     );end rodAddMPPChopHole
     => t / nil
```

Description

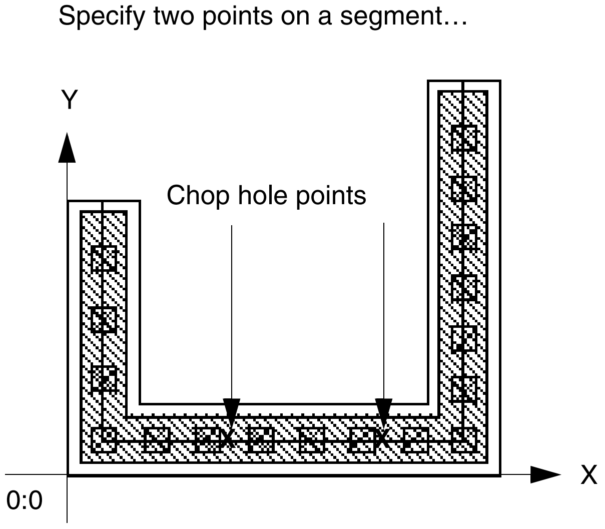
Adds a chop hole to all choppable parts of a ROD multipart path. To define a chop hole, you specify a list of points on the centerline of the master path. The results depend on whether the master path is choppable or not.

Master path is not choppable, but subparts are choppable.

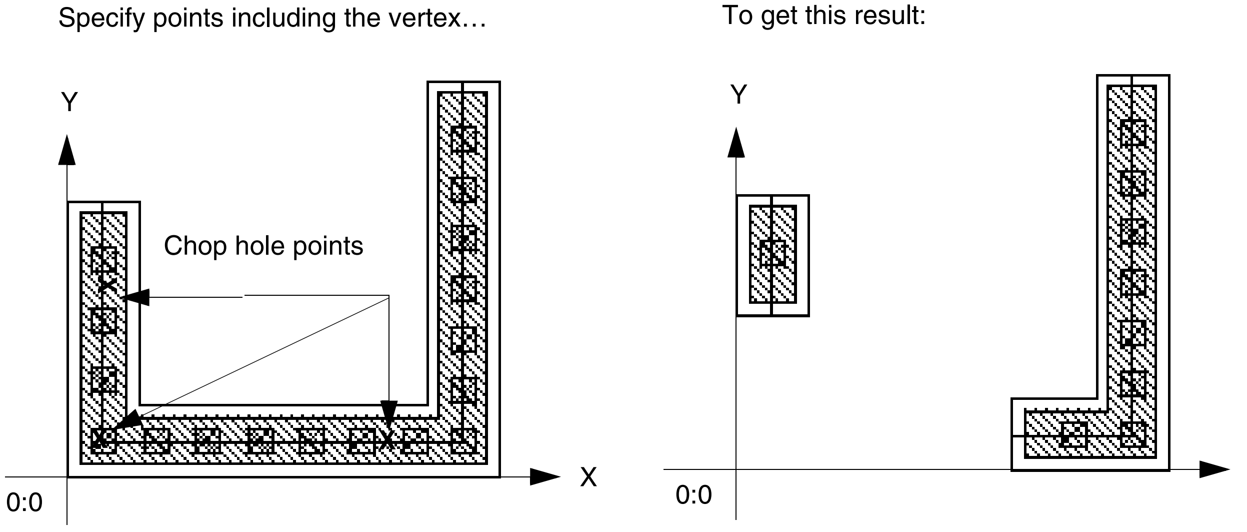
Master path is choppable; therefore all subparts are choppable.



To add a chop hole to a segment of a multipart path, you must specify two points on the same segment. If you specify only one point on a segment, the system ignores the point.



To add a chop hole that includes a vertex (the intersection of two segments), you must include the vertex point in your list.



For more information about how chop holes affect multipart paths, see Creating Paths with rodCreatePath in the *Virtuoso Relative Object Design User Guide*.

Arguments

R_rodObj	IROD object ID associated with the multipart path. Default: none

`l_pts`	List of two or more points on the centerline of the master path specifying where to create a chop hole. The chop hole affects choppable parts only. You are required to specify at least two points. The system discards duplicate and collinear points but does not consider them an error.
	Use one of the following formats:
	`list(` *`x`*`:`*`y`* *`x`*`:`*`y`* `... )`
	or
	`list( list(`*`x y`*`) list(`*`x y`*`) ... )`
	Default: none

Value Returned

`t`	The multipart path was chopped successfully.
`nil`	An error occurred and the multipart path was not chopped.

rodAddPoints

```
rodAddPoints(
     l_point1
     l_point2
     );end rodAddPoints
     => l_point / nil
```

Description

Adds two points together and returns the resulting point as *`l_point`*.

Arguments

`l_point1`	A set of coordinates in one of the following formats: *`x`*`:`*`y`* or `list(`*`x y`*`)` Default: none
`l_point2`	A set of points in one of the following formats: *`x`*`:`*`y`* or `list(`*`x y`*`)` Default: none

Value Returned

`l_point`	The point that results from adding *`l_point1`* and *`l_point2`*.
`nil`	No points were added due to an error.

rodAddSubPart

```
rodAddSubPart(
     R_rodObjId
     t_subpartType
     l_subpartParamList
     );end rodAddSubPart
     => R_rodSubpartId / nil
```

Description

Creates a new subpart of the type specified by `subpartType` with the given list of parameters (`subpartParamList`) and adds it to an existing multipart path (MPP).

Arguments

`R_rodObjId`	ROD object ID associated with the multipart path (MPP).
`t_subpartType`	String specifying the subpart type. It can be an offset, enclosure, or a rectangle subpart.
`l_subpartParamList`	A list of parameters similar to MPP parameter list used in `rodCreatePath`. This list is specified as `l_offsetSubpathArgs`, `l_encSubpathArgs`, or `l_subrectArgs` for MPPs. For more information, see `multipartPathTemplates` n the *Virtuoso Technology File Data: ASCII Files Reference* Manual.

Value Returned

`R_rodSubpartId`	Returns *`R_rodSubpartId`* if the subpart was created and added successfully to the MPP.
`nil`	The subpart was not created or added to the MPP.

Example

```
rodAddSubPart(mppId "offset" list(list("poly1" "drawing") 1.0 t 0.0 "center" 0.1
0.1))
```

rodAddToX

```
rodAddToX(
     l_point
     n_num
     );end rodAddToX
     => l_point / nil
```

Description

Adds a number to the X coordinate of *`l_point`* and returns the resulting point as *`l_point`*.

Arguments

`l_point`	A set of coordinates in one of the following formats: *`x`*`:`*`y`* or `list(`*`x y`*`)` Default: none
`n_num`	Signed integer or floating-point number. Default: none

Value Returned

`l_point`	The point resulting from adding *`n_num`* to the X coordinate of *`l_point`*.
`nil`	No number was added due to an error.

rodAddToY

```
rodAddToY(
     l_point
     n_num
     );end rodAddToY
     => l_point / nil
```

Description

Adds a number to the Y coordinate of *l_point* and returns the resulting point as *l_point*.

Arguments

l_point	A set of coordinates in one of the following formats: *x*:*y* or list(*x y*) Default: none
n_num	Signed integer or floating-point number. Default: none

Value Returned

l_point	The point resulting from adding *n_num* to the Y coordinate of *l_point*.
nil	No number was added due to an error.

rodAlign

```
rodAlign(
     [ ?alignObj R_alignObj ]
     [ ?alignHandle S_alignHandle ]
     [ ?refObj R_refObj ]
     [ ?refHandle S_refHandle ]
     [ ?refPoint l_refPoint ]
     [ ?maintain g_maintain ]
     [ ?xSep txf_xSep ]
     [ ?ySep txf_ySep ]
     );end rodAlign
     => R_rodObj / nil
```

Description

Aligns a named object by a point handle on that object to a specific point or to a point handle on a reference object. You can align objects that are at different levels of hierarchy as long as both objects are in the same top-level layout cellview. You can specify positive or negative separation between alignment points in the direction of both the X and Y axes, either as absolute distances or with Cadence SKILL language expressions. The system applies the offset from the reference point or reference object to the object to be aligned.

Arguments

`?alignObj` *`R_alignObj`*

ROD object ID associated with the object you want to align. Use with *`S_alignHandle`* to specify a point on *`R_alignObj`* to use for alignment. The object you want to align must exist in the same top-level cellview as the reference object or reference point.
Default: none

`?alignHandle` *`S_alignHandle`*

Symbol or character string specifying the name of the handle to use as the alignment point on *`R_alignObj`*.
Enclose character strings in quotation marks (""); precede symbols with a single quotation mark ('). ROD converts symbols to character strings.
Default: `centerCenter`

`?refObj` *`R_refObj`*

ROD object ID associated with the object to be used as a reference for the alignment. The reference object must exist in the same top-level cellview as the object specified by *`R_alignObj`*.

If you do not specify *`l_refPoint`*, this argument is required. If you also specify *`l_refPoint`*, the value of *`l_refPoint`* overrides the values of *`R_refObj`* and *`S_refHandle.`*
Default: none

`?refHandle` *`S_refHandle`*

Symbol or character string specifying the name of the handle on the reference object to which you want to align *`R_alignObj`*. If you also specify *`l_refPoint`*, it overrides the value of *`R_refObj`* and *`S_refHandle.`*
Default: `centerCenter`

`?refPoint` *`l_refPoint`*

A list of two coordinates specifying the point to which you want to align *`R_alignObj`*. Use one of the following formats:
`x``:`*`y`*
or
`list(` *`x y`* `)`
If you do not specify *`S_refHandle`*, *`R_refObj`*, or *`l_refPoint`*, then *`l_refPoint`* defaults to `0:0`. If you specify *`l_refPoint`* and also specify *`R_refObj`* (with or without specifying *`S_refHandle`*), the value of *`l_refPoint`* overrides the values of *`R_refObj`* and *`S_refHandle.`*
Default: `0:0`

`?maintain` *`g_maintain`*

A Boolean value specifying whether or not to maintain an alignment after it is created. The value must be `t` or `nil`.

When set to `t`, moving one of two aligned objects moves the other with it. When set to `nil`, moving one of two aligned objects does not move the other object.

You can use the `nil` setting to create temporary alignments for the purpose of arranging objects in relation to each other. These alignments are no longer in effect after the `rodAlign` function completes execution.
Default: `t`

`?xSep` *`txf_xSep`*	An integer, floating-point number, or Cadence SKILL language expression specifying the vector distance in the direction of the X axis from either the reference point handle *`S_refHandle`* on *`R_refObj`* or the reference point *`l_refPoint`* to the alignment point handle *`S_alignHandle`* on *`R_alignObj`*. The value can be positive or negative or a SKILL expression that evaluates to a positive or negative number. SKILL expressions are reevaluated whenever the design is opened, the technology file is updated, or one of the aligned objects is moved or modified. Default: `0`
`?ySep` *`txf_ySep`*	An integer, floating-point number, or Cadence SKILL language expression specifying the distance along the Y axis from either the reference point handle *`S_refHandle`* on *`R_refObj`* or the reference point *`l_refPoint`* to the alignment point handle *`S_alignHandle`* on *`R_alignObj`*. The value can be positive or negative or a Cadence SKILL language expression that evaluates to a positive or negative number. SKILL expressions are reevaluated whenever the design is opened, the technology file is updated, or one of the aligned objects is moved or modified. Default: `0`

Value Returned

`R_rodObj`	ROD object ID for the ROD object information associated with *`R_alignObj`*.
`nil`	An error occurred and no objects were aligned.

Additional Information

When you align an object to a point, the system creates a user-defined handle for the point with the system-assigned name `handle`*`n`*, where *`n`* is a number that makes the handle name unique in the cellview. For information about user-defined handle, see "User-Defined Handles" in the *Virtuoso Relative Object Design User Guide*.

You can also use the `rodAlign` function to quickly arrange objects in relation to each other by temporarily aligning them. When the *`g_maintain`* argument is set to `nil`, alignments are no longer maintained after the function is executed.

For an overview about aligning objects, see "Aligning Objects" in the *Virtuoso Relative Object Design User Guide*. The following examples illustrate aligning ROD objects.

Figure 1-1 on page 33 shows the object `polyRect` before and after it is aligned to the reference point 0:0.

Figure 1-1 Aligning a Rectangle to a Point

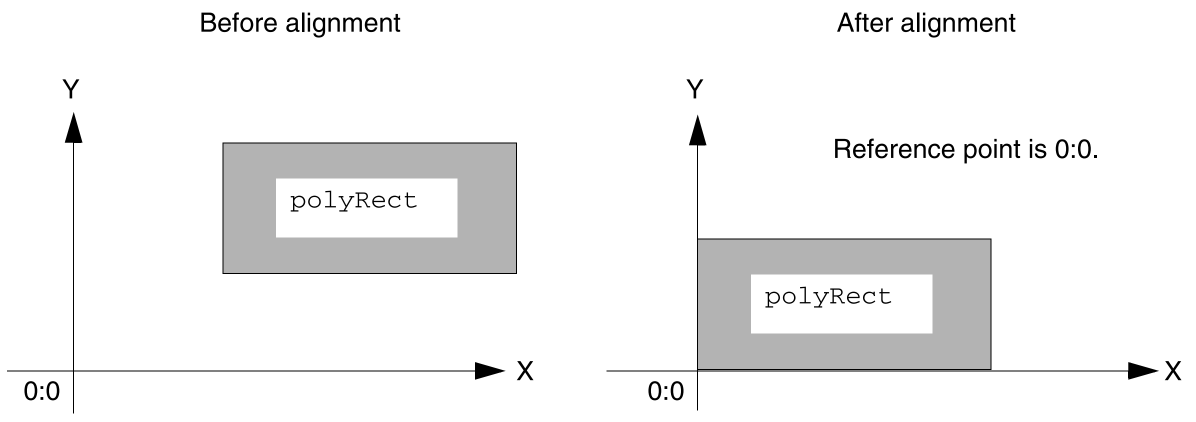


Figure 1-2 on page 34 shows the objects `polyRect` and `rect0` both before and after `polyRect` is aligned to `rect0`. After alignment, the `centerLeft` handle of `polyRect` is aligned to the `centerRight` handle of `rect0`, with a positive separation along the X axis. The object `polyRect` is on the right side of `rect0` because the system applies the offset from the reference object `rect0` to the aligned object `polyRect`.

Figure 1-2 Aligning Two Rectangles with Positive Separation

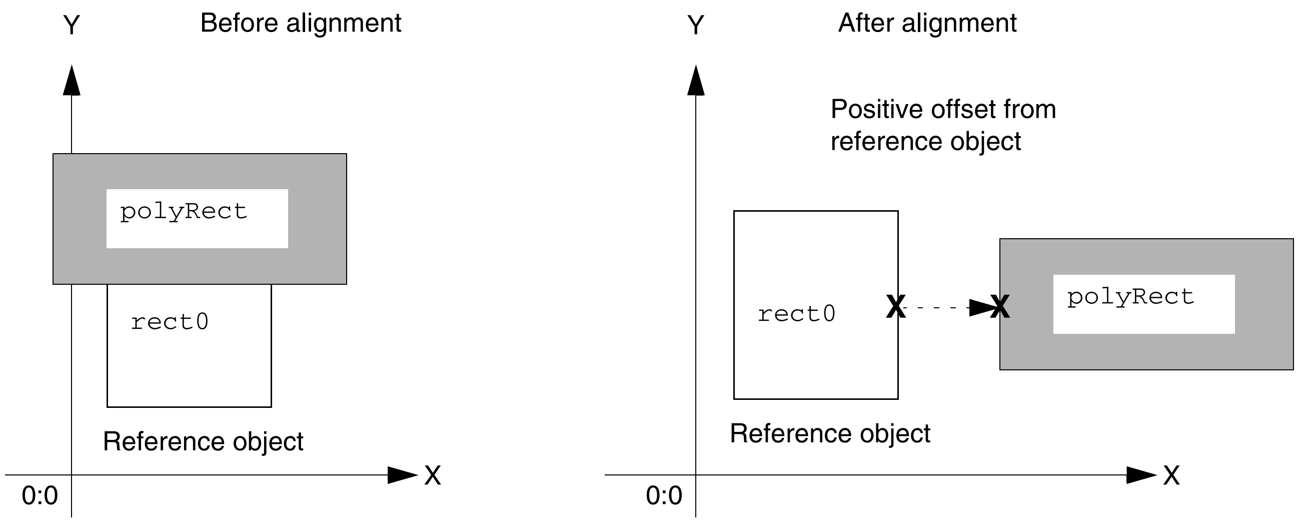
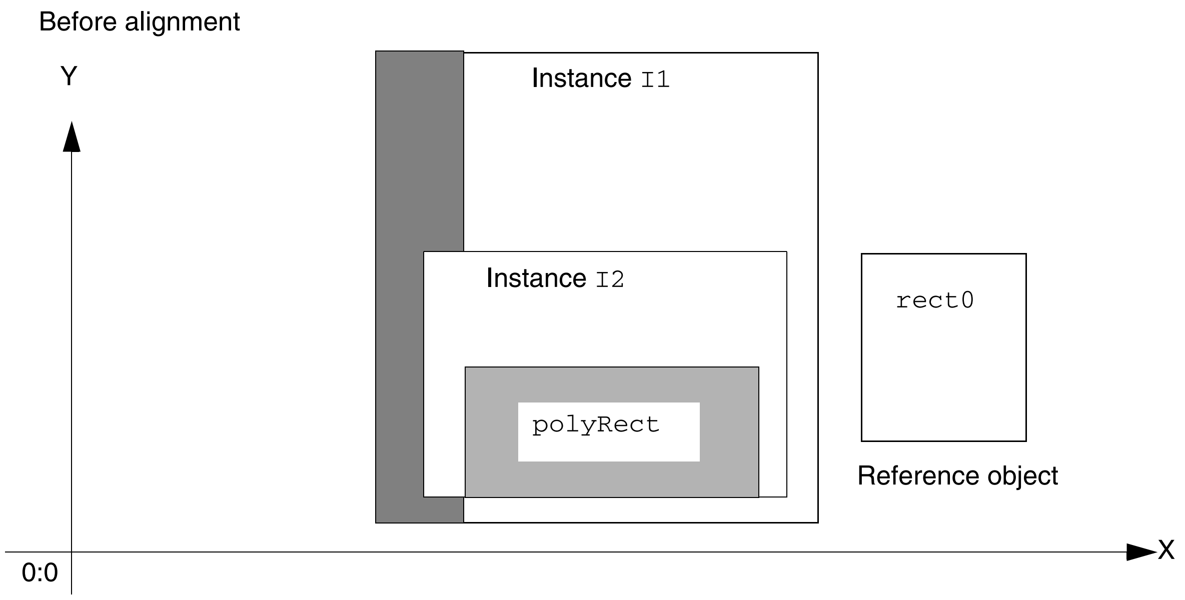


Figure 1-3 on page 34 shows instance `I1` and the rectangle `rect0`. Instance `I1` contains instance `I2`, which contains the object `polyRect`.

Figure 1-3 Aligning an Instance and Rectangle Across Hierarchy



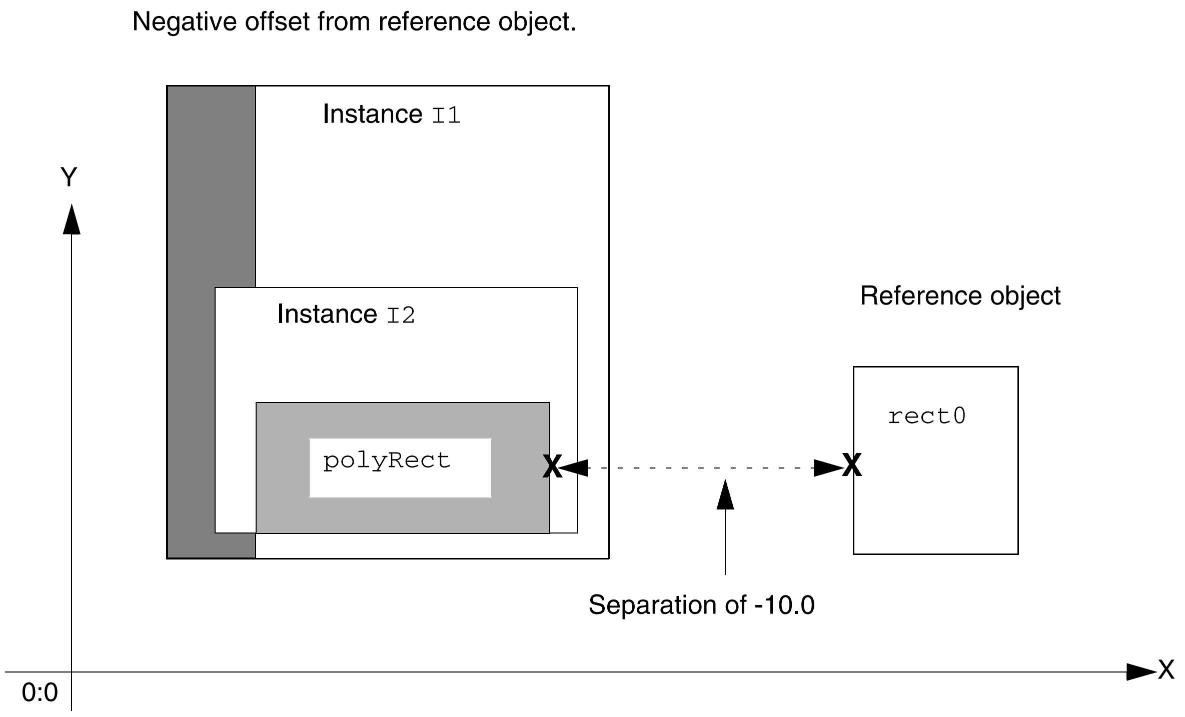
The hierarchical name for `polyRect` is `I1/I2/polyRect`.

Figure 1-4 on page 35 shows instance I1 and `rect0` after alignment across hierarchy. `rect0` is the reference object. The `centerRight` handle of

`I1/I2/polyRect` is aligned to the `centerLeft` handle of `rect0`, with a -10 separation in the direction of the X axis.

Because the offset is negative along the X axis, the aligned object `I1/I2/polyRect` is on the left side of the reference object `rect0`. Subsequently, if you move `rect0`, instance `I1` and everything in it moves with `rect0`, maintaining the correct alignment; also, if you move instance `I1`, `rect0` moves with it.

Figure 1-4 Instance and Rectangle After Alignment



Note: For information about how the Virtuoso layout editor *Maintain Connections* option works for aligned objects, see "Maintaining Connections for ROD Objects" in the *Virtuoso Relative Object Design User Guide*.

Querying an Object for Alignment

You can query any named object to see what the object is aligned to. When you query a hierarchical object, the system displays all top-level alignments for the object. To see the alignments for an object, you can use the Virtuoso layout editor Edit Properties command (the ROD section) or type commands in the CIW, as shown in the examples below.

As shown in the example, `I1/I2/polyRect` is aligned to `rect0`. To query `rect0` in Figure 1-4 on page 35 for alignments, type the following in the CIW:

```
cvId = deGetCellView()
rodGetObj( "rect0" cvId )~>align
```

The system displays text similar to the following in the CIW:

```
((rodObj:2585108 "cR" rodObj:2585114 "cL" -10.0
        0.0
    )
)
```

The text above states that the `centerRight` (`cR`) handle on `I1/I2/polyRect` (`rodObj:2585108`) is aligned to the `centerLeft` (`cL`) handle on `rect0` (`rodObj:2585114`), with a separation of -10.0 along the X axis and no separation along the Y axis.

Moving Aligned Objects

When you move a ROD object that is aligned to one or more ROD objects, the aligned objects also move. When you move a ROD object that is aligned to a reference point, the system changes the value of the reference point handle to keep the point in the same relationship with the object.

rodAssignHandleToParameter

```
rodAssignHandleToParameter(
     [ ?parameter S_parameter ]
     [ ?rodObj R_rodObj ]
     [ ?handleName Sl_handleName ]
     [ ?displayName S_displayName ]
     [ ?displayExpression S_displayExpression ]
     [ ?stretchDir S_stretchDir ]
     [ ?stretchType S_stretchType ]
     [ ?moveOrigin g_moveOrigin ]
     [ ?updateIncrement f_updateIncrement ]
     [ ?userData g_userData ]
     [ ?userFunction Sl_userFunction ]
     [ ?stretchKey S_stretchKey ]
     [ ?shape S_shape ]
     );end rodAssignHandleToParameter
     => t / nil
```

Description

Assigns one or more user- or system-defined point handles on a ROD object within a SKILL-based parameterized cell (Pcell) to a single Pcell parameter, so you can change the value of the parameter by graphically stretching the handle(s). Stretching a handle has the same effect as changing the value of the associated parameter for the Pcell instance using the Edit Properties form. You cannot stretch handles on the Pcell master. All `rodAssignHandleToParameter` statements **must** occur within the body-of-code section of a SKILL-based Pcell.

For an overview of stretchable Pcells, see "Stretchable Parameterized Cells" in the *Virtuoso Relative Object Design User Guide*.

Arguments

`?parameter` *`S_parameter`*

Symbol or character string specifying the name of the Pcell parameter to which you want to assign one or more point handles. The data type of the Pcell parameter must be an integer or floating-point number, unless you also specify the *`Sl_userFunction`* argument.

This argument is required. If you do not specify this argument or the parameter does not exist in the parameter declaration section of the Pcell code, the function reports an error.
Valid Values: any parameter associated with the Pcell
Default: none

`?rodObj` *`R_rodObj`*

ROD object ID identifying the object in a Pcell that has the point handles you want to assign to a parameter. This argument is required. If you do not specify this argument or the specified ROD object ID is not valid, the function reports an error.
Default: none

`?handleName` *`Sl_handleName`*

Symbol or character string specifying the name of a single user- or system-defined point handle or a list of character strings or symbols specifying the names of one or more point handles. You can specify any point handles that are associated with the object specified by *`R_rodObj`*.

This argument is required. If you do not specify this argument or a specified handle is not a point handle for the specified object, the function reports an error.
Default: none

`?displayName` *`S_displayName`*

Symbol or character string specifying any text that you want the system to display next to the Pcell while you stretch it, such as the name of a parameter or an expression. For example, the following are valid values: `"length"`, `'length`, `"length + 2"`, and `"My Handle"`.

If you also specify *`S_displayExpression`*, the system displays both its value “as is”, without evaluating it, and the result of evaluating the *`S_displayExpression`* argument, separated by the string `" = "`, in the following format:

`displayName` = *`displayExpression_evaluation_result`*

The following examples show what the system displays when you specify both *`display`* arguments:

`S_display Name`	*`S_display Expr`*	*`displayExpr`* **Evals To**	**What You See**
`"length"`	`"length * 2"`	10	length = 10
`"length"`	`"width + 3"`	16	length = 16

If you do not specify *`S_displayExpression`*, the system treats *`S_displayName`* as an expression and evaluates it dynamically as you stretch the Pcell. The system displays both the string or symbol and the result of the evaluation, separated by the string `" = "`, in the following format:

`displayName = displayName_evaluation_result`

If *`S_displayName`* does not evaluate successfully, the system displays `Eval not successful` in the layout cellview instead of a value. If no value is specified for *`S_displayName`*, nothing is displayed for it.

For more information about specifying this argument, see “Additional Information” on page 47.
Default: none

`?displayExpression` *`S_displayExpression`*

Symbol or character string specifying an expression, such as a parameter name or equation. For example, the following are valid values: `"width"`, `'width`, and `"width + 2"`.

The system evaluates the value of this argument dynamically as you stretch the Pcell and displays the result next to the Pcell. When you specify this argument but do not specify the *`S_displayName`* argument, the system displays the result of the evaluation in the following format:

```
result_of_evaluation
```

For example, for the expression `"width"`, where `width` equals 12, the system displays `12.0`; for the expression `"width + 2"`, the system displays `14.0`.

If you also specify *`S_displayName`*, the system displays both the value of *`S_displayName`*, without evaluating it, and the result of evaluating the *`S_displayExpression`* argument, separated by the string `" = "`, in the format:

```
string_or_symbol =
        displayExpression_evaluation_result
```

For more information about specifying this argument, see "Additional Information" on page 47.
Default: none

`?stretchDir` *`S_stretchDir`*

Symbol or character string specifying the axis along which you want the Pcell to stretch. Although you can stretch the Pcell in any direction, the system computes the change to *`S_parameter`* only in the direction of the axis specified by this argument; the system updates the associated Pcell parameter with a change in the direction of either the X or Y axis.
Valid Values: `x`, `X`, `y`, or `Y`
Default: `x`

`?stretchType` *`S_stretchType`*

Symbol or character string specifying whether the stretch is relative to the position of the handle or to the center of the Pcell. When the value is `absolute`, the stretch is relative to the handle; when the value is `relative`, the stretch is relative to the center of the Pcell.

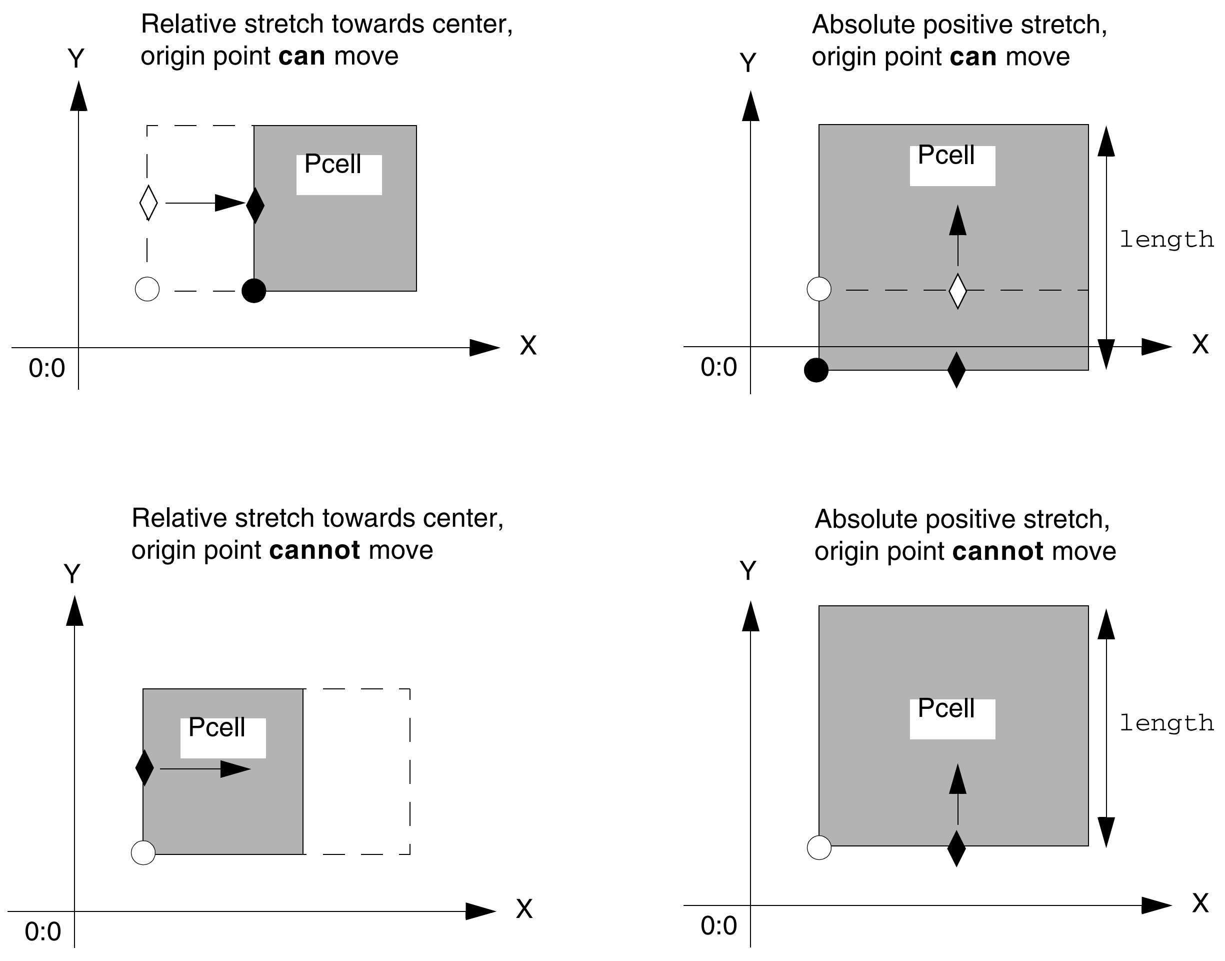
For an `absolute` stretch, an upward stretch away from the position of the handle (in a positive direction along the axis) is an increment; a downward stretch away from the position of the handle (in a negative direction along the axis) is a decrement. For a `relative` stretch, a stretch is away from the center of the Pcell is an increment; a stretch towards the center of the Pcell is a decrement.

For examples showing relative vs. absolute, see “Results of Stretching a Handle” in the *Virtuoso Relative Object Design User Guide*.
Valid Values: `relative` or `absolute`
Default: `relative`

`?moveOrigin` *`g_moveOrigin`*

Boolean value specifying whether the system can move the origin point of an instance during the stretch of a handle on the instance or on an object in the instance. When the value is `t`, the origin point can move during a stretch; when the value is `nil`, the origin point cannot move during a stretch. Specifying `t` is most useful when *`S_stretchType`* is set to `relative` and the location of the stretch handle coincides with and affects the bounding box of the Pcell instance.

When the origin point can move, the edges of the instance adjacent to the origin point can move also, which allows handles on those edges to move in the direction of the stretch.


Relative stretch towards center, origin point **can** move
Y
Pcell
X
0:0
Absolute positive stretch, origin point **can** move
Y
Pcell
length
X
0:0
Relative stretch towards center, origin point **cannot** move
Y
Pcell
X
0:0
Absolute positive stretch, origin point **cannot** move
Y
Pcell
length
X
0:0

When the origin cannot move, the edges of the instance adjacent to the origin point cannot move either, which prevents handles on those edges from moving in the direction of the stretch.
For more examples, see "Results of Stretching a Handle" in the *Virtuoso Relative Object Design User Guide*.
Valid Values: `t`, `nil`
Default: `nil`

`?updateIncrement` *`f_updateIncrement`*

Floating-point number specifying how often the system updates the Pcell parameter (*`S_parameter`*) and regenerates the Pcell during a stretch operation. If you specify a negative number, the system uses the default. The default is the value of the layout editor environment variable `udpatePcellIncrement`. If you specify *`f_updateIncrement`*, its value overrides the value of `updatePCellIncrement`.
Default: value of the layout editor environment variable `udpatePcellIncrement`

`?userData` *`g_userData`*

User data of any data type. The system evaluates the data and includes the result as part of the *`r_defstruct`* structure that is passed to *`Sl_userFunction`* as input. This argument lets you pass the values of variables and constants defined in your Pcell code to a user-defined function. For example, you can pass the values of design rules, such as `minSpacing` and `minWidth`.

`?userFunction` *`Sl_userFunction`*

Symbol, string, or lambda function specifying a user-defined function. A lambda function lets you define a function without using a function name. The purpose of user-defined functions is to calculate new values for Pcell parameters. Do not use user-defined functions for other purposes, such as to create shapes. User-defined functions are executed interactively while the Pcell is stretched; therefore, a user-defined function cannot depend on Pcell arguments.

Note: It is recommended not to use lambda functions when using the Express Pcell feature. Instead, a normal function should be defined and passed as a function object to ROD for stretch handling. This is because in the Express Pcell feature, the Pcell code is evaluated and the submaster is stored in cache to avoid reevaluation. However, the lambda functions are temporary functions that need to be evaluated each time and therefore, cannot be stored in cache as a function object pointer. Therefore, any storage of the lambda functions as a function object becomes stale for the next session.

As input to the user-defined function, the system passes a defstruct that is predefined by the ROD software. A defstruct is a named SKILL structure containing a collection of one or more slots for variables. For more information about defstructs, see the `defstruct` function in the *SKILL Language Reference*. In this syntax, the defstruct is referred to as *`r_defstruct`*. Replace *`defstruct`* with your own name, for example, `myDefstruct`.

This destruct contains the following predefined variable slots:

- *t_handleName*
  Text specifying the name of the handle currently being stretched.
- *n_increment*
  An integer or floating-point number specifying the increment or decrement as a result of stretching *t_handleName*.
- *d_origInstId*
  Database ID of the original instance being stretched.
- *g_parameters*
  Lets you set the value of a Pcell parameter, using the following construct:
  ```
  myDefstruct->parameters->parameter_name =
                                  parameter_value
  ```
  where *parameter_value* must evaluate to the same data type as *parameter_name*. The *g_parameters* variable is write-only; it is not preloaded with parameter names and values. After the user-defined function completes, the system looks at *g_parameters* within the predefined defstruct, and if any parameters were set, the system uses those values as the new parameter values.
- *g_paramVal*
  Current value of the Pcell parameter named *S_parameter*; the value can have any data type.
- *R_rodObj*
  ROD object ID identifying the original object whose handle is being stretched.
- *S_stretchDir*
  String or symbol specifying the stretch direction for the current handle being stretched (*t_handleName*).
- *d_stretchMaster*
  Interim database ID of the most recent Pcell submaster in virtual memory.
- *g_userData*
  Result of the evaluation of the *g_userData* argument by the system, when user data is specified.

The system replaces the value of *`S1_parameter`* with the value returned. If the value for *`S1_parameter`* is also set using the `myDefstruct->parameters->` construct, the construct value overrides the value returned by the user-defined function.

User-defined functions **must** return a value with the same data type as *`S_parameter`*, and the value assigned by a `myDefstruct->parameters->` construct must have the same data type as the parameter. If a value does not have the same data type as its parameter, the system issues a warning and assigns zero or a null value to the parameter; the evaluation of the Pcell could fail.

For more information about the ROD predefined destruct, see "The r_defstruct Structure" on page 53. For more information about user-defined functions, see "User-Defined Functions" on page 51.

Default: none

`?stretchKey` *`S_stretchKey`*

Symbol or character string that you can specify as per your requirement. The internal representation of this string is allocated dynamically. It is a null string, until a key is specified in the `rodAssignHandleToParameter` call.

Default: null

`?shape` *`S_shape`*

Symbol or character string that you can use to control the default shape of the stretch handle.

Valid Values: `diamond`, `square`, `dot`

Value Returned

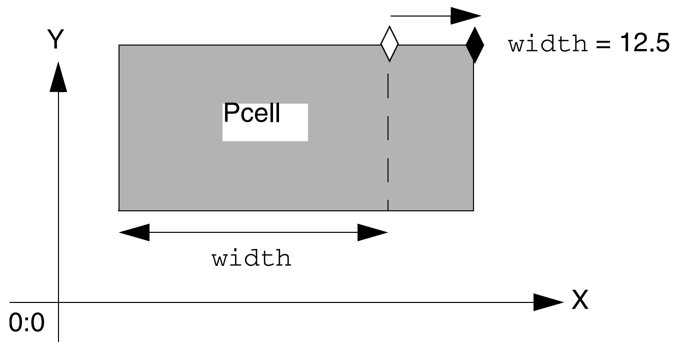
`t`	Handles were successfully assigned to the Pcell parameter.
`nil`	An error occurred and no handle was assigned.

Additional Information

Displaying Parameter Information

You can define information to be displayed for a Pcell parameter by specifying the *`S_displayName`* and *`S_displayExpression`* arguments. For *`S_displayExpression`*, you can include any Pcell parameter in the expression. The system displays the information next to the upper-right corner of the Pcell when you select the assigned stretch handle and updates the information as you drag the handle.

For example, for a Pcell that contains only a rectangle and has one parameter, `width`, you might want to display the name of the parameter and its new value as you stretch the Pcell.



Depending on the information you want to display, specify the arguments as follows:

- To display

  `displayName = displayName_evaluation_result`

  specify only the *`S_displayName`* argument. The system displays its contents as an unevaluated text string on the left and the result of evaluating the *`S_displayName`* argument on the right, separated by the string `" = "`.

  The following examples show what the system displays when you specify only the *`S_displayName`* argument:

S_displayName	**`Name`** **Evals To**	**What You See**
`` `length ``	10	`length = 10`

S_displayName	**`Name` Evals To**	**What You See**
`"length * 2"`	20	`length * 2 = 20`
`"onOffSwitch"`	`t`	`onOffSwitch = t`
`"M"`	evaluation fails	M = Eval not successful

If no value is specified for *`S_displayName`*, nothing is displayed for it.

- To display

  `displayExpression_evaluation_result`

  specify only the *`S_displayExpression`* argument. You can use any Pcell parameter in the expression. The system evaluates the expression and displays the result.

  For example, if *`S_displayExpression`* contains the expression `width + 3`, and the variable `width` is currently equal to `5.0`, the system displays the following: `8.0`

- To display

  `displayName = displayExpression_evaluation_result`

  specify both the *`S_displayName`* and *`S_displayExpression`* arguments. The system displays the contents of *`S_displayName`* as an unevaluated text string on the left and the result of evaluating *`S_displayExpression`* on the right, separated by the string `" = "`.

  The following examples show what the system displays when you specify both *`display`* arguments:

S_displayName	S_displayExpr	*`displayExpr`* Evals To	What You See
`"length"`	`"length * 2"`	10	length = 10
`"length"`	`"width + 3"`	16	length = 16

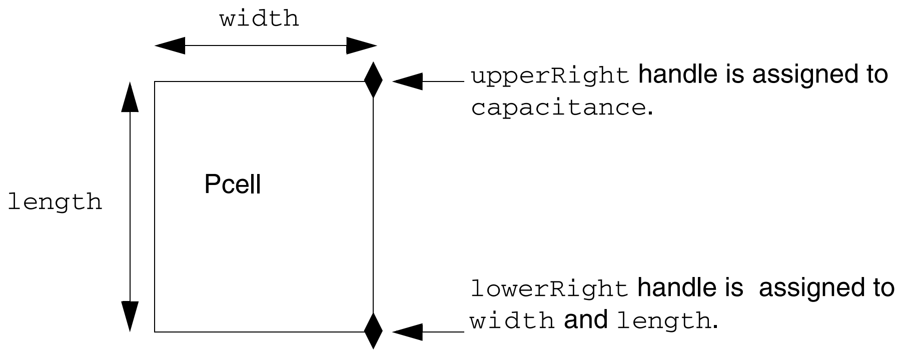
Examples of Display Information

In the following example, the Pcell contains only a rectangle, with handles assigned to the three Pcell parameters: `capacitance`, `width`, and `length`. The Pcell code contains three `rodAssignHandleToParameter` statements:

- One `rodAssignHandleToParameter` statement assigns the `upperRight` handle to the `capacitance` parameter.

- A second `rodAssignHandleToParameter` statement assigns the `lowerRight` handle to the `width` parameter.
- A third `rodAssignHandleToParameter` statement assigns the `lowerRight` handle to the `length` parameter.

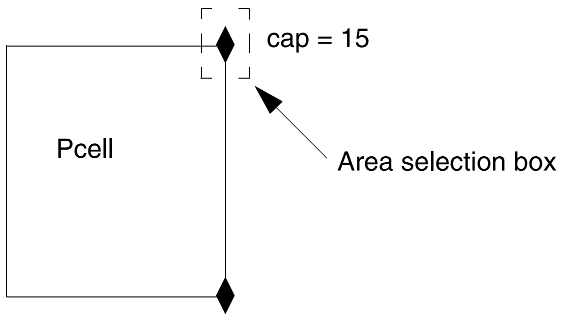
  In each statement, the information to be displayed is specified as follows:


width
length
Pcell
upperRight handle is assigned to capacitance.
lowerRight handle is assigned to width and length.

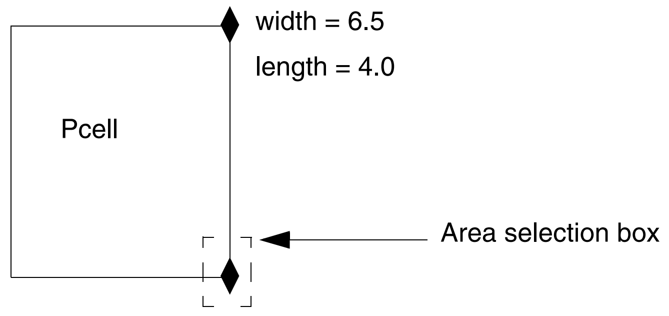
- For the `capacitance` parameter, `S_displayName` is `cap` and `S_displayExpression` is the expression

  `width*length/capCoeff`
- For the `width` parameter, `S_displayName` is `width` and `S_displayExpression` is not specified.
- For the `length` parameter, `S_displayName` is `length` and `S_displayExpression` is not specified.

When you select the `upperRight` handle, you see the following information displayed:



When you select the `lowerRight` handle, you see the following information displayed:



For a table showing examples of how to specify the display arguments, see Table 1-2 on page 50.

The following table shows the results of specifying one or both of the *`S_display`* arguments.

Table 1-2 Examples of Specifying the Display Arguments

S_displayName	S_display Expression	Evaluates To	What You See
`'length`		10	length = 10
`"length * 2"`		20	length * 2 = 20
`"onOffSwitch"`		`t`	onOffSwitch = `t`
`"My Handle"`		evaluation fails	My Handle = Eval not successful
	`"length"`	10	10
	`"length * 2"`	20	20
	`"onOffSwitch"`	`t`	`t`
	`"myHandle"`	evaluation fails	Eval not successful
`"length"`	`"length * 2"`	20	length = 20
`"length"`	`"width + 3"`	16	length = 16
`"On-Off Switch"`	`"onOffSwitch"`	`t`	On-Off Switch = `t`
`"My Handle"`	`"myHandle"`	evaluation fails	My Handle = Eval not successful

User-Defined Functions

When you assign a handle to a parameter, you can define your own function (`Sl_userFunction`) to calculate the value of the Pcell parameter (`S_parameter`) to which you are assigning the handle. These functions are executed when the Pcell is stretched; they cannot depend on Pcell parameters.

Note: A user-defined function should be used only to calculate a new value for a Pcell parameter. Do not use user-defined functions for other purposes, such as creating objects.

Information Passed to a User-Defined Function

As input to user-defined functions, the ROD software passes a predefined, user-named defstruct (`r_defstruct`). For a description of the contents of the defstruct, see the `Sl_userFunction` argument description.

Specifying Values for Parameters Using the -> Operator

Within a user-defined function, you can use the following construct to set the value for any Pcell parameter:

```
myDefstruct->parameters->parameter_name = parameter_value
```

where `myDefstruct` is your name for the defstruct and `parameter_value` must be the same data type or evaluate to the same data type as `parameter_name`. For example,

```
myDefstruct->parameters->width = 6.0
```

The system stores all such constructs in the place-holder variable `g_parameters` within the `r_defstruct` structure.

Note: The `g_parameters` variable is not preloaded with parameters and their values.

What the System Does

The system executes the user-defined function, which calculates and returns a new value for `S_parameter`. After the user-defined function completes, the system looks at the variable `g_parameters` within the `r_defstruct` structure and, if values were set for any parameters, uses those values as the new values for the parameters.

Note: If you set the value for `S_parameter` using the `myDefstruct->parameters->` construct, the system uses the value assigned by the construct rather than the value returned by the user-defined function.

Loading User-Defined Functions

When there are user-defined functions specified, the system calls them at each drag increment during the stretch and when you complete the stretch operation. You need to make sure that user-defined functions are loaded into the system before the system needs them. User-defined functions must be loaded prior to dragging a stretch handle belonging to a Pcell.

The safest way to make sure that user-defined functions are always available when needed is to attach them to your library. Then the system automatically loads them when the library is opened.

Follow the steps below to attach a source file for a user-defined function to a library:

1. Place the source file within the library directory.
2. Create a file named `libInit.il` in the library directory.
3. The system automatically loads the `libInit.il` file when the library is opened.
4. Within the `libInit.il` file, write a `load` statement like this:

```
load(
    strcat(
            ddGetObjReadPath( ddGetObj( "libName" ))
            "/filename.il"
    ) ; end strcat
) ; end load
```

Note: Avoid using graphical SKILL functions such as `hiSetBindKey` in your `libInit.il` file; doing so will result in an error. For more information see “What to Avoid in the libInit.il File” in the *Virtuoso Parameterized Cell Reference*.

Lambda Functions

Using a *lambda function* lets you define a function without a name, in the form of a list. This is useful for writing temporary or local functions.

The syntax for a lambda function is:

```
lambda(
     (l_formalArguments)
     g_expr1 ...)
     => U_result)
     ); end lambda
```

Arguments

`lambda`	Identifies the following arguments as belonging to an unnamed function.
`l_formalArguments`	List of one or more variable names.

`g_expr1`...	The remaining elements in the list are SKILL expressions that are evaluated when the `lambda` function is called.

For example, the following lambda function defines an unnamed function capable of computing the length of the diagonal side of a right-angled triangle:

```
lambda( (x y) (sqrt (x*x + y*y))
```

When `x = 3` and `y = 4`, the value returned is `5.0`.

You do not need to precede a lambda function with a single quotation mark.

For more information about lambda functions, see "Basic Concepts" in Chapter 3 of the *SKILL Language User Guide* and "`lambda`" in the *SKILL Language Reference*.

The r_defstruct Structure

`r_defstruct` is a user-named structure that is predefined by the ROD software to contain a series of slots for specific variables. A defstruct behaves just like a disembodied property list. The system passes the defstruct as input to user-defined functions defined within the `rodAssignHandleToParameter` function. For more information about defstructs, see `defstruct` in the *SKILL Language Reference*.

You can access the value of the variables in the *`r_defstruct`* structure by using the name you assigned to the defstruct, the structure access operator (`->`) and the name of the variable. For example, to access the value of the variable *`R_rodObj`*, type the following:

```
myDefstruct->rodObj
```

Examples

The following examples use user-defined functions.

Example 1

The following user-defined function adds the value of *`n_increment`* to *`g_paramVal`*, prints the current values of the `myDefstruct` variable slots, displays a warning message if the return value is less than zero, and returns the new value of *`g_paramVal`*.

```
procedure( exampleUserFunction(myDefstruct)
    let(((returnVal myDefstruct->paramVal + myDefstruct->increment))
         ;; print all parameter values:
        printf("myDefstruct->handleName = %s\n", myDefstruct->handleName)
        printf("myDefstruct->increment = %f\n", myDefstruct->increment)
        printf("myDefstruct->origInstId = %L\n", myDefstruct->origInstId)
        printf("myDefstruct->parameter = %s\n", myDefstruct->parameter)
        printf("myDefstruct->paramVal = %L\n", myDefstruct->paramVal)
        printf("myDefstruct->rodObj = %L\n", myDefstruct->rodObj)
        printf("myDefstruct->stretchDir = %s\n", myDefstruct->stretchDir)
        printf("myDefstruct->stretchMaster = %L\n\n",
                myDefstruct->stretchMaster)
        if( (returnVal <= 0.000001)
            progn(warn("returnVal for parameter %s must be > 0\n"
                        myDefstruct->parameter)

              ) ; end of progn
        ) ; end if
        returnVal
    ) ; end of let
) ; end or procedure
```

The user-defined function produces the following output:

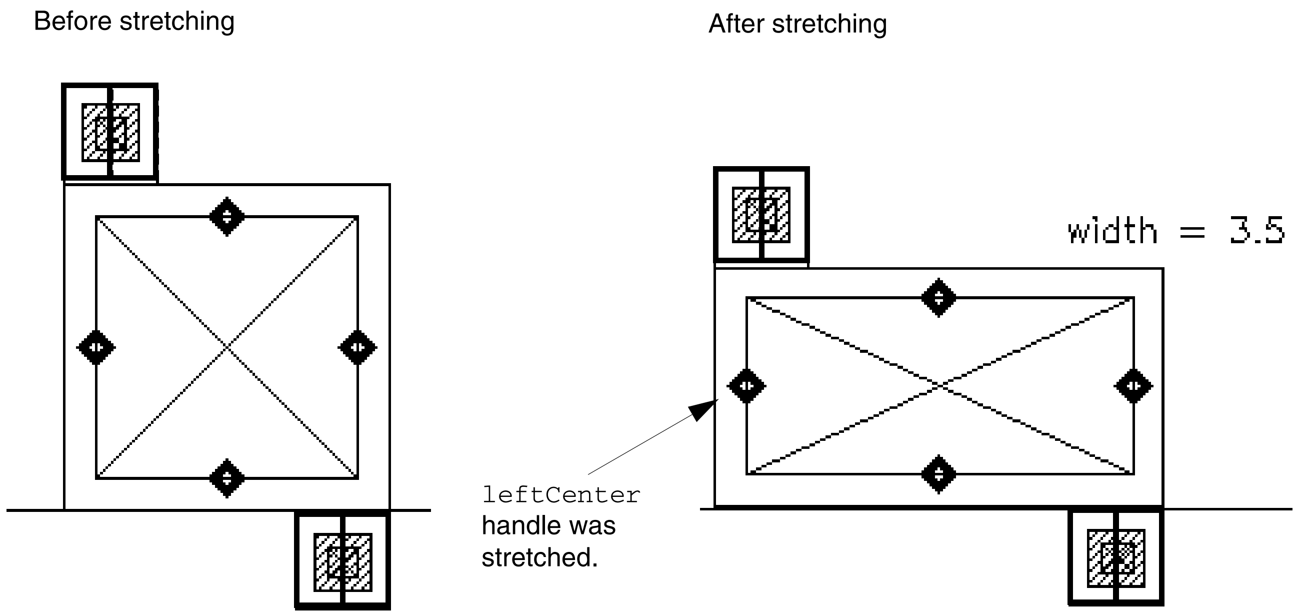
```
myDefstruct->handleName = upperRight
myDefstruct->increment = 0.300000
myDefstruct->origInstId = db:67200100
myDefstruct->parameter = length
myDefstruct->paramVal = 1.0
myDefstruct->rodObj = rodObj:66969624
myDefstruct->stretchDir = y
myDefstruct->stretchMaster = db:67195948
```

Example 2

The following example shows how to specify the *g_userData* argument and the *g_parameters* element of the *r_defstruct* structure for user-defined functions.

In the example, a capacitor has a fixed capacitance parameter (`cap`) and variable `width` and `length` parameters. As `length` is stretched, the system automatically adjusts `width` to maintain the fixed capacitance value; the converse is also true.

The figure below shows what the capacitor looks like before and after stretching the `leftCenter` handle to increase `width`. The system decreases `length` to keep the capacitance constant.



To make the Pcell process independent, design rules such as `minSpacing` were retrieved from the technology file and passed to the user-defined function with the optional *`g_userData`* argument.

In the code below, each section does the following:

- Section 1 is the part of the Pcell code that gets the technology file ID and uses it to retrieve the technology rules used in the input to the user-defined function in Section 3.
- Section 2 is the part of the Pcell code that assigns the `leftCenter` and `rightCenter` handles to the `width` parameter.
- Section 3 is a user-defined function that is called from within the Pcell, but which is not part of the Pcell code. The procedure updates `length` when `width` is stretched. The *`g_userData`* argument passes technology rule values to the user-defined function.

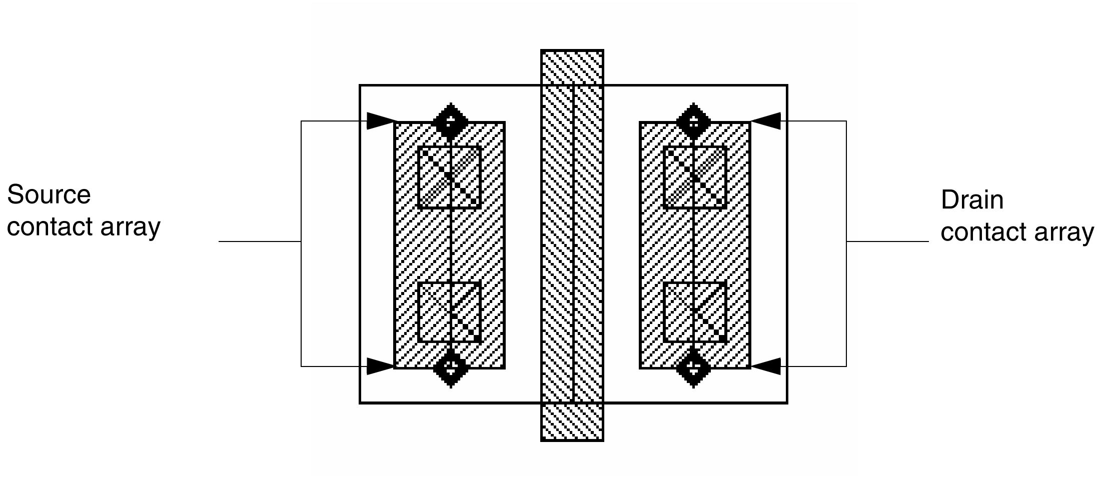
As a SKILL Pcell author, you can write a process-independent Pcell by using design rules from your technology file, such as `minSpacing` and `minEnclosure`. The *`g_userData`* parameter lets you pass values for constants and variables, such as the values of technology rules, directly to your user-defined function.

```
;; SECTION 1: Pcell code that gets minimum values from technology file
tfId = techGetTechFile( pcCellView )
capCoeff = techGetParam( tfId "capCoeff" )
;layer1 and layer2 are top and bottom layers for capacitor
 lay1Ovia = techGetOrderedSpacingRule( tfId "minEnclosure"
                                 layer1 "cont")
 lay2Ovia = techGetOrderedSpacingRule( tfId "minEnclosure"
                                 layer2 "cont" )
 viaW = techGetSpacingRule( tfId "minWidth""cont" )
 grid = techGetMfgGridResolution( tfId )
 minCap = techGetParam( tfId "minCap" )
;; SECTION 2: Pcell code that assigns handles to parameters
rodAssignHandleToParameter(
    ?parameter "width"
    ?rodObj cnt
    ?displayName "width"
    ?handleName list("leftCenter" "rightCenter")
    ?stretchType "absolute"
    ?stretchDir "x"
    ?userData list(list(capCoeff lay1Ovia lay2Ovia viaW grid
                 minCap ))
    ?userFunction "myfunc"
) ; end rodAssignHandleToParameter
;; SECTION 3:User-defined function called from within pcell.
;; The procedure updates the length when width is changed by
;; the stretch operation.
procedure(myfunc(myDefstruct)
    let((returnVal mylist capCoeff lay1Ovia lay2Ovia viaW grid
           minCap)
        returnVal = myDefstruct->paramVal + myDefstruct->increment
        mylist = car(myDefstruct->userData)
        capCoeff = nth(0 mylist)
        lay1Ovia = nth(1 mylist)
        lay2Ovia = nth(2 mylist)
        viaW = nth(3 mylist)
        grid = nth(4 mylist)
        minCap = nth(5 mylist)
        cap = (myDefstruct->stretchMaster)~>parameters~>cap
        if(lay1Ovia > lay2Ovia then
            minCapWidth = (2 * lay1Ovia + viaW)
        else
           minCapWidth = (2 * lay2Ovia + viaW)
         ); end if
        ;; Calculate the current length
        length = spcRound((cap/capCoeff/returnVal) grid)
        when(length < minCapWidth
            progn(warn("returnVal for parameter %s must be >
                     minCapWidth %L\n" "length" minCapWidth))
            length = minCapWidth
            returnVal = spcRound((cap/capCoeff/length) grid)
        ); end when
        ;; Reset the length so the capacitance value remains fixed.
        myDefstruct->parameters->length = length
        returnVal
    ); end let
); end procedure
```

For more information about structures, see “Defstructs” in Chapter 4 of the *SKILL Language User Guide* and the `defstruct` function in Chapter 2 of the *SKILL Language Reference*.

Example 3

The following example shows a MOS transistor Pcell with stretch handles on the source and drain contact arrays. The MOS transistor Pcell calls two user-defined functions.



The MOS transistor Pcell parameters are specified as follows:

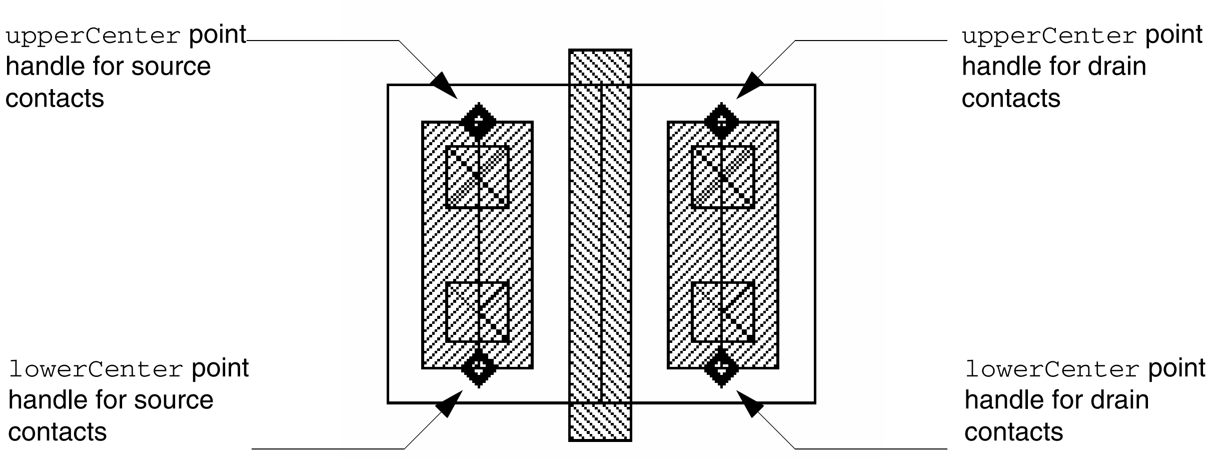
Parameter Name	Default Value	Parameter Description
`w`	1.3	Width of gate
`l`	0.25	Length of gate
`leftcov`	1.0	Left contact coverage
`rightcov`	1.0	Right contact coverage
`leftPos`	"top"	Position of left contacts
`rightPos`	"top"	Position of right contacts

The contact arrays are aligned to the gate. For example, when `leftPos` equals `top`, the `upperLeft` point handle of the gate is aligned to the `upperRight` point handle of the left contact array, with an offset along both the X and Y axes.

The stretch handles are assigned as follows:

- To make the source contacts stretchable, the `upperCenter` and `lowerCenter` point handles on the source contact array are assigned to the Pcell parameter `leftcov` with the stretch type `relative` and the stretch direction `Y`.

- To make the drain contacts stretchable, the `upperCenter` and `lowerCenter` point handles on the drain contact array are assigned to the Pcell parameter `rightcov` with the stretch type `relative` and the stretch direction `Y`.


upperCenter point handle for source contacts
upperCenter point handle for drain contacts
lowerCenter point handle for source contacts
lowerCenter point handle for drain contacts

Two user-defined functions control the number of contacts generated and whether they are aligned to the top or bottom of the diffusion:

- The user-defined function `contcov` sets a range for the contacts from a 100% coverage to the minimum contact width for the layer from the technology file.
- The user-defined function `myStretch` identifies the handle currently being stretched. It determines whether the contacts are aligned to the top or bottom of the diffusion.

To look at the code for the Pcell and the user-defined functions, see "Stretchable MOS Transistor" in the *Virtuoso Relative Object Design User Guide.*

Example 4: Stretching the Sample MOS Transistor

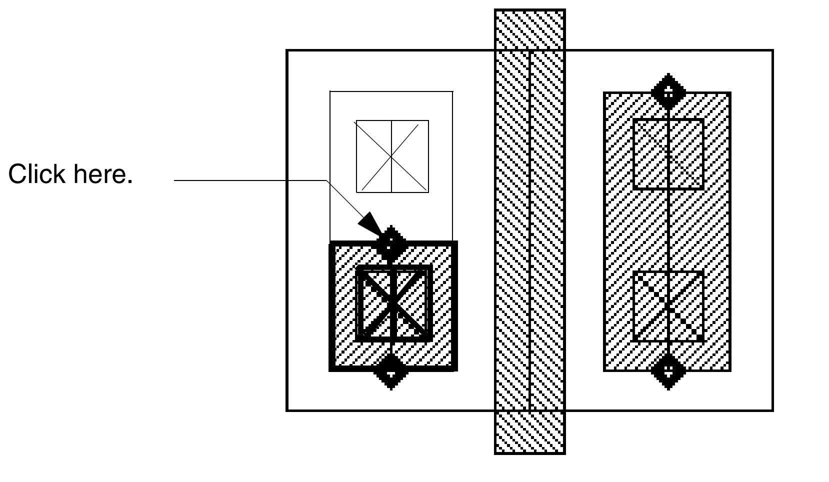
You can stretch the MOS transistor shown in Example 3 in a negative direction towards the Y axis to reduce the number of contacts in the array to one, using the Virtuoso layout editor *Stretch* command.

1. Choose *Edit – Stretch*.
2. Select the `upperCenter` stretch handle on the left contact array using an area-selection box.
3. To specify the reference point, click the `upperCenter` stretch handle on the left contact array.

   Notice that the following information appears to the right of the Pcell:

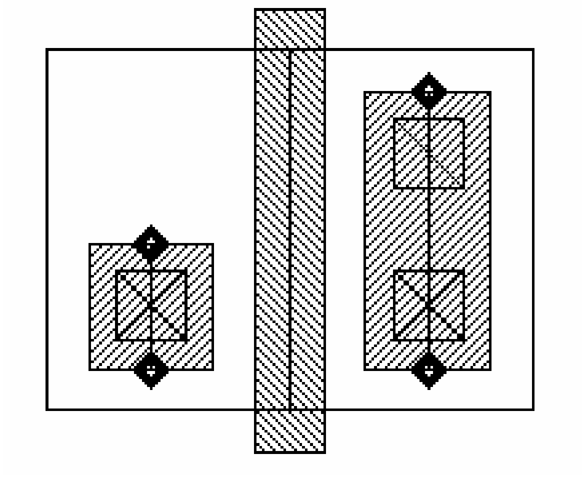
```
leftcov = 1
```

4. To specify the new location, click above the bottom transistor.



Note: If you have trouble selecting the stretch handle, try turning off the *Gravity On* option on the Layout Editor Options form.

The MOS transistor Pcell instance now looks like this:



rodCheck

```
rodCheck(
     d_cvID
     [ g_createMarkers ]
     ); end rodCheck
     => Rl_rodObj
```

Description

Searches the specified cellview for multipart paths (MPPs) for which the geometry does not match the definition information, and returns a list of the ROD object IDs. The MPPs found by this function include those that have lost their subshape definition information. By default, this function places warning markers over the potentially affected MPPs. (If you do not want to see warning markers, set the *`g_createMarkers`* argument to `nil`.) Although subparts with missing definition information are still visible graphically, their definitions have been deleted from the database. Editing an affected MPP causes an immediate loss of the subpart graphics.

Arguments

`d_cvID`	Database ID for the cellview you want to search. Default: none
`g_createMarkers`	Boolean value indicating whether or not to create warning markers for the potentially affected MPPs found by the `rodCheck` function. Valid values: `t`, `nil` Default: `t`

Value Returned

`Rl_rodObj`	ROD object ID or list of ROD object IDs that identifies the multipart paths for which the geometry does not match the definition information.

rodCheckAllMPPsInCellView

```
rodCheckAllMPPsInCellView(
     t_libName
     t_cellName
     t_viewName
     g_fix
     g_createMarkers
     ); end rodCheckAllMPPsInCellView
     => l_defectiveMPPIds / nil
```

Description

Finds and opens the design specified by `t_libName`, `t_cellName`, and `t_viewName`, and checks if any MPP in the design contains defective internal data. If MPPs containing defective internal data are found, `rodCheckAllMPPsInCellView` fixes them and creates markers on them depending on the values of `g_fix` and `g_createMarkers` arguments.

Arguments

t_libName	Character string representing the library name of the design
t_cellName	Character string representing the cell name of the design
t_viewName	Character string representing the view name of the design
g_fix	Boolean value indicating whether or not to fix the potentially affected MPPs found by `rodCheckAllMPPsInCellView`. Valid values: `t`, `nil`
g_createMarkers	Boolean value indicating whether or not to create warning markers for the potentially affected MPPs found by the `rodCheckAllMPPsInCellView` function. Valid values: `t`, `nil`

Value Returned

l_defectiveMPPIds	ROD object ID or list of ROD object IDs that identifies the multipart paths containing defective internal data.
`nil`	Returns `nil` if any of the arguments are invalid or the MPPs could not be fixed.

Possible outputs

g_fix	**g_createMarkers**	**Output**
`t`	`nil`	Fix but do not create markers
`t`	`t`	Fix and create markers
`nil`	`nil`	Do not fix or create markers. Returns the list of defective MPP IDs.
`nil`	`t`	Do not fix, but create markers.

rodCheckMPPs

```
rodCheckMPPs(
     d_cellViewId
     R_rodObj | list(R_rodObj ...)
     g_fix
     g_createMarkers
     );end rodCheckMPPs
     => l_defectiveMPPIds / nil
```

Description

Checks if the given list of multipart paths (MPPs) contains corrupt internal data. If MPPs containing defective internal data are found, `rodCheckMPPs` fixes or creates markers to highlight them, or does both.

Arguments

d_cellViewId	ID of the cellview that contains the MPP.
R_rodObj / list(r_rodObj...)	
	A single `rodObjectId` of an MPP or a list of `rodObjectIds` of MPPs.
g_fix	Boolean value indicating whether or not to fix the potentially affected MPPs found by `rodCheckMPPs`. Valid values: `t`, `nil`
g_createMarkers	Boolean value indicating whether or not to create warning markers for the potentially affected MPPs found by the `rodCheckMPPs` function. Valid values: `t`, `nil`

Value Returned

l_defectiveMPPIds	Returns the list of `rodObjectIds` of MPPs containing defective internal data among the list provided.
`nil`	Returns `nil` if any of the arguments are invalid or the MPPs could not be fixed.

Possible outputs

g_fix	g_createMarkers	Output
`t`	`nil`	Fix but do not create markers
`t`	`t`	Fix and create markers
`nil`	`nil`	Do not fix or create markers. Returns the list of defective MPPIds.
`nil`	`t`	Do not fix, but create markers.

rodCreateHandle

```
rodCreateHandle(
     [ ?name S_name ]
     [ ?type S_type ]
     [ ?value g_value ]
     [ ?rodObj R_rodObj ]
     );end rodCreateHandle
     => t / nil
```

Description

Creates a user-defined handle for any ROD object (named shape, instance, or cellview); the object must be at level zero in the hierarchy. For example, you can create a handle for `polyRect` or for instance `I1`, but not for `I1/polyRect`.

To create a handle for a cellview, you must first get the ROD object ID for the cellview with the `rodGetObj` function.

After you create a handle, you can access it through hierarchy using the `rodGetHandle` function and the database access operator (`~>`). The `rodGetHandle` function automatically transforms (converts) the coordinates of the ROD object into the coordinate system of the top-most cellview.

Arguments

`?name` *`S_name`*

Symbol or character string specifying a unique name for the new handle. Enclose character strings in quotation marks (""); precede symbols with a single quotation mark ('). ROD converts symbols to character strings.

Do not use existing user- or system-defined handle names. For example, you can assign the name `myPoint`, but not `centerCenter`. If you do not assign a name, then the system assigns a name unique within the cellview, as follows: `handle0`, `handle1`, `handle2`, etc.

Valid Values: any symbol or character string except existing user- or system-defined handle names
Default: unique name assigned by the system

`?type` *`S_type`*

Symbol or character string specifying the type of data stored at the handle. The type can be Boolean, floating point, integer, point, string, or a Cadence SKILL language expression. This argument is required.
Valid Values: `boolean`, `float`, `int`, `point`, `string`, `ILExpr`
Default: none

`?value` *`g_value`*

Value assigned to the handle. The value must evaluate to match the data type defined by *`S_type`*. When *`S_type`* is Boolean, the value must be `t` or `nil`. When *`S_type`* is `string`, the value must be either a character string or a symbol. This argument is required.
Valid Values: a value that matches the data type specified by *`S_type`*; when *`S_type`* is Boolean, `t` or `nil`.
Default: none

`?rodObj` *`R_rodObj`*

ROD object ID associated with the object for which you want to create a user-defined handle. This argument is required.
Default: none

Value Returned

`t`	Handle created successfully.
`nil`	An error occurred and no handle was created.

rodCreatePath

```
rodCreatePath(
     [ ?name S_name]
     [ ?layer txl_layer ]
     [ ?width n_width ]
     [ ?pts l_pts ]
     [ ?justification S_justification ]
     [ ?offset n_offset ]
     [ ?endType S_endType ]
     [ ?beginExt n_beginExt ]
     [ ?endExt n_endExt ]
     [ ?choppable g_choppable ]
     [ ?cvId d_cvId ]
     [ ?fromObj Rl_fromObj ]
     [ ?size txf_size ]
     [ ?startHandle l_startHandle ]
     [ ?endHandle l_endHandle ]
     [ ?prop l_prop ]

     [rodConnectivityArgs...  ]
          ;start ROD Connectivity Arguments (rodConnectivityArgs)
          [ ?netName S_netName ]
          [ ?termName S_termName ]
          [ ?termIOType S_termIOType ]
          [ ?pin g_pin ]
          [ ?pinAccessDir tl_pinAccessDir ]
          [ ?pinLabel g_pinLabel ]
          [ ?pinLabelHeight n_pinLabelHeight ]
          [ ?pinLabelLayer txl_pinLabelLayer ]
          [ ?pinLabelFont S_pinLabelFont ]
          [ ?pinLabelDrafting g_pinLabelDrafting ]
          [ ?pinLabelOrient S_pinLabelOrient ]
          [ ?pinLabelOffsetPoint l_pinLabelOffsetPoint ]
          [ ?pinLabelJust S_pinLabelJust ]
          [ ?pinLabelRefHandle S_pinLabelRefHandle ]
          ;end ROD Connectivity Arguments

     [ ?offsetSubPath            l_offsetSubpathArgs... ]
     ;start Offset Subpath Arguments (l_offsetSubpathArgs)
     list(
          list(
               [ ?layer txl_layer ]
               [ ?width n_width ]
               [ ?choppable g_choppable ]
               [ ?sep n_sep ]
               [ ?justification S_justification ]
               [ ?beginOffset n_beginOffset ]
               [ ?endOffset n_endOffset ]
               [ ?prop l_prop ]
               ;repeat ROD Connectivity Arguments here
          );end first offset subpath list
               ...
          );end offset subpath lists
     ;end Offset Subpath Arguments

     [ ?encSubPath        l_encSubpathArgs...  ]
     ;start Enclosure Subpath Arguments (l_encSubpathArgs)
     list(
          list(
               [ ?layer txl_layer ]
               [ ?enclosure n_enclosure ]
               [ ?choppable g_choppable ]
               [ ?beginOffset n_beginOffset ]
               [ ?endOffset n_endOffset ]
               [ ?prop l_prop ]
               ;repeat ROD Connectivity Arguments here
          );end first enclosure subpath list
               ...
          ;end of enclosure subpath lists
     ;end Enclosure Subpath Arguments

     [ ?subRect l_subrectArgs...  ]
     ;start Subrectangle Arguments (l_subrectArgs)
     list(
          list(
               [ ?layer txl_layer ]
               [ ?width n_width ]
               [ ?length n_length ]
               [ ?choppable g_choppable ]
               [ ?sep n_sep ]
               [ ?justification S_justification ]
               [ ?space n_space ]
               [ ?beginOffset n_beginOffset ]
               [ ?endOffset n_endOffset ]
               [ ?beginSegOffset n_beginSegOffset ]
               [ ?endSegOffset n_endSegOffset ]
               [ ?prop l_prop ]
               [ ?gap S_gap ]
               ;repeat ROD Connectivity Arguments here
               [ ?diagonal g_diagonalSubRect  ]
          );end first subrectangle list
                ...
          ;end subrectangle lists
       );end Subrectangle Arguments
     );end rodCreatePath

     => R_rodObj / nil
```

Description

Creates a path consisting of one or more parts at level zero in the hierarchy on the same or on different layers from a list of points or from one or more existing objects. A path consisting of multiple parts is called a *multipart path*. You can use the `rodCreatePath` function to create one-part paths, simple multipart paths, or complex multipart paths such as guard rings, transistors, buses, and shielded paths. You can assign one or more property names and values to a multipart path.

Note: `rodCreatePath` is a core database SKILL function that checks only for even multiples of database units to create a path. As a result, the created path can be off grid. To create a path with required constraints and additional level of automation use an end-application SKILL function, such as `leCreatePath`.

A multipart path consists of a single *master path* and one or more *subparts*. The master path is an ordinary path; however, it is the defining part of a multipart path; all subparts are based on the master path. You can assign a property name and value, or a list of property names and values, to the master path and/or to each subpart. The subparts can be any or all of the following: offset subpaths, enclosure subpaths, and sets of subrectangles.

You can create any number of subparts. All subparts exist in relation to and depend on the master path. A subrectangle in a set of subrectangles is not an individual shape; it is part of that specific set of subrectangles. In the Virtuoso layout editor, you cannot select or edit an individual subrectangle. Subrectangles are created on grid.

Note: Although you cannot select an individual subrectangle, it is possible to obtain the database ID for an individual subrectangle. If you use the database ID to change connectivity information (such as terminal name) for one or more individual subrectangles, your change is applied to the individual subrectangles immediately, and propagated to all of the subrectangles in the set of subrectangles when the MPP is modified in any way (such as moved).

For a detailed overview of multipart paths, see “Multipart Paths”.

You can also create MPPs and save them as templates in your technology file by using the graphical user interface or by editing the ASCII version of your technology file. For information about using the graphical user interface, see “Creating and Editing Multipart Paths” in the *Virtuoso Layout Suite L User Guide*. For information about editing your technology file, see “`multiPartPathtcCreatreTemplates`” in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

The following topics are discussed in this section, following the argument descriptions:

- Creating Self-Intersecting Paths
- Specifying Arguments as nil

- Formatting List-of-Lists Arguments for Subparts
- How the System Follows to Create Subrectangles
- Disconnecting Shapes in a Multipart Path

Arguments

Master Path Arguments

`?name` *`S_name`*	Symbol or character string specifying the name for the master path. Enclose character strings in quotation marks (""); precede symbols with a single quotation mark ('). ROD converts symbols to character strings. The name must be unique for the cellview. If you do not specify a name, the system assigns a unique name, consisting of the prefix `path`, followed by a number. For example, for the first master path in a cellview for which you do not specify a name, the system assigns the name `path0`, if unique; for the second, `path1`, and so on. Default: `path`*`n`*

`?layer` *`txl_layer`*	Text string, integer, or list specifying the layer or layer-purpose pair for the master path, such as
	`?layer "metal1"`
	`?layer 45`
	`?layer list("metal1" "drawing")`
	
	Enclose string values in quotation marks. Do not use a symbol. For lists, use the following format:
	`list(` *`layer purpose`* `)`
	
	Examples of layer-purpose pairs:
	`?layer list("metal1" "drawing")`
	`?layer list(45 252)`
	`?layer list(45 "drawing")`
	`?layer list("metal1" 252)`
	
	You are required to specify a layer.
	Default Purpose: `drawing`

`?width` *`n_width`*	Positive integer or floating-point number specifying the width of the master path. If you do not specify the width, the system uses the `minWidth` rule for the master path layer from the technology file. If the `minWidth` rule is not defined in the technology file, the `rodCreatePath` function reports an error. Default: the `minWidth` rule for the master path layer from the technology file

`?pts` *`l_pts`*	List of points defining the centerline of the master path. Specify a point for the start and end of the first segment, and specify a point for the end of subsequent segments. You must specify at least two noncoincident points. You cannot create self-intersecting master paths. You are required to specify a list of points. The system discards duplicate and collinear points before creating the path but does not consider them an error.
	
	Use one of the following formats:
	`list(` *`x`*`:`*`y x`*`:`*`y`* `... )`
	or
	`list( list(`*`x y`*`) list(`*`x y`*`) ... )`
	
	You must specify either *`l_pts`* or *`Rl_fromObj`*. If you specify both, the system ignores the *`l_pts`* argument.
	Default: none

`?justification` *`S_justification`*	
	Symbol or character string specifying the part of the master path to offset from the point list: centerline, left edge, or right edge. Justification is relative to the direction of the path. Valid Values: `center`, `left`, `right` Default: `center`
`?offset` *`n_offset`*	Signed integer or floating-point number specifying the distance by which the master path edge or centerline, depending on *`S_justification`*, is offset from the point list specified by the *`l_pts`* argument. A positive value creates the master path to the left of the point list; a negative value creates the master path to the right of the point list. For a detailed description about offsetting the master path, see "Master Paths" in the *Virtuoso Relative Object Design User Guide*. Default: `0`
`?endType` *`S_endType`*	

Symbol or character string specifying the type of ends of the master path.
Valid Values: `flush`, `offset`, `octagon`, `variable`
Default: `flush`

`?beginExt` *`n_beginExt`*

Zero or positive integer or floating-point number specifying the distance by which the starting end of the master path extends beyond its first point. This argument is ignored unless *`S_endType`* is set to `variable`.
Default: `0`

`?endExt` *`n_endExt`*

Zero or positive integer or floating-point number specifying the distance by which the ending end of the master path extends beyond its last point. This argument is ignored unless *`S_endType`* is set to `variable`.
Default: `0`

`?choppable` *`g_choppable`*

Boolean value indicating whether or not a ROD path can be chopped. The value must be `t` or `nil`. When a path has subparts and the master path is choppable, all subpaths and sets of subrectangles must be choppable also. When a path has subparts and the master path is not choppable, each subpath and/or set of subrectangles can be choppable or not.
Default: `t`

`?cdId` *`d_cvId`*

Database ID for the cellview in which you are creating a ROD path. If `rodCreatePath` occurs in the body of a `pcDefinePCell` function or `tcCreateCDSDeviceClass` function call, the default value is `pcCellView` or `tcCellView`, respectively; otherwise, this argument is required.
Default: none

`?fromObj` *`Rl_fromObj`*

ROD object ID or list of ROD object IDs identifying a named object or list of named objects that you want to use as a source for creating a new ROD path. The source objects can be instances, rectangles, polygons, paths, lines, dots, labels, and/or text-display objects. You must specify either the *`Rl_fromObj`* argument or the *`l_pts`* argument. If you specify both, the system ignores the *`l_pts`* argument.
Default: none

`?size` *`txf_size`*

A signed integer, floating-point number, or Cadence SKILL language expression specifying the distance between a bounding box around the source object (*`Rl_fromObj`*) and the centerline of the generated master path. When the source is more than one object, *`txf_size`* specifies the distance between a bounding box around all of the objects and the centerline of the master path.

By default, the generated path has four segments and forms a ring. A positive number creates a path that is longer than the circumference of the bounding box; a negative number creates a path that is shorter than the circumference of the bounding box.

When you specify a SKILL expression, it must evaluate to a positive or negative integer or floating-point number. SKILL expressions are evaluated only when the new path is created.

If you specify a source object (*`Rl_fromObj`*) but do not specify *`txf_size`*, then *`txf_size`* defaults to `0.0`. If you specify *`txf_size`*, you must also specify *`Rl_fromObj`*, or the system reports an error.
Default: `0.0`

`?startHandle` *`S_startHandle`*

Symbol or character string specifying the starting point for the generated path, such as `start`*`n`*, where *`n`* is the segment number.
Valid Values: `start`*`n`*, `mid`*`n`*, `end`*`n`*, `startLast`, `midLast`, `endLast`
Default: `start0`

`?endHandle` *`S_endHandle`*

Symbol or character string specifying the ending point for the generated path, such as `start`*`n`*, where *`n`* is the segment number

Valid Values: `start`*`n`*, `mid`*`n`*, `end`*`n`*, `startLast`, `midLast`, `endLast`
Default: `start0`

`?prop` *`l_prop`*

A list or lists of lists, where each list contains the name of a property and its value. When the property name is a string, enclose it in quotes. For a description of the valid data types, see the "About the Add Property Form" in the *Virtuoso Layout Suite L User Guide*. For Boolean properties, the value must be `t` or `nil`. For example, you would specify a single Boolean property as follows:

```
?prop list("myProp" t)
```

and a list of properties as follows:

```
?prop list(
   list("myProp1" propValue1)
   list("myProp2" propValue2)
   ...
   list("myPropN" propValueN)
)
```

For a more detailed description about specifying lists of lists, see Formatting List-of-Lists Arguments for Subparts.
Default: `nil`

`rodConnectivityArgs...`

A list containing rod connectivity arguments. For more details on these arguments, see ROD Connectivity Arguments (rodConnectivityArgs).

`l_offsetSubpathArgs...`

A list containing one or more lists, each of which specifies the arguments for one offset subpath. The subpath is created in relationship to the master path. For information about how to specify lists of lists, see Formatting List-of-Lists Arguments for Subparts.

For more information on *`l_encSubPathArgs`* arguments, see Enclosure Subpath Arguments (l_encSubpathArgs).

`l_encSubPathArgs...`

A list containing one or more lists, each of which specifies the arguments for one enclosure subpath. The enclosure subpath is created in relationship to the centerline of the master path. For information about how to specify lists of lists, see Formatting List-of-Lists Arguments for Subparts.

For more information on *`l_offsetSubpathArgs`*, see Enclosure Subpath Arguments (l_encSubpathArgs).

`l_subRectArgs...`	A list containing one or more lists, each of which specifies the arguments for one set of unnamed subrectangles. The set of subrectangles is created in relationship to the centerline of the master path. For information about how to specify lists of lists, see Formatting List-of-Lists Arguments for Subparts. For more information on *`l_subRectArgs`*, see Subrectangle Arguments (l_subrectArgs).

ROD Connectivity Arguments (rodConnectivityArgs)

`?netname` *`S_netName`*

Symbol or character string specifying the name of the net with which you want to associate the shape. Enclose character strings in quotation marks (""); precede symbols with a single quotation mark (').
ROD converts symbols to character strings. If the net does not exist, the system creates it, using the name specified by *`S_netName`*. To associate the shape with a net, this argument is required. If you do not want to specify this argument, omit it or specify its value as `nil`.
Default: `nil`

`?termName` *`S_termName`*

Symbol or character string specifying the name of the terminal and net with which you want to associate the shape. Enclose character strings in quotation marks (""); precede symbols with a single quotation mark (').
ROD converts symbols to character strings. If the terminal and/or net does not exist, the system creates it, using the name specified by *`S_termName`*.

If you specify both *S_termName* and *S_netName*, their values must be the same. If a terminal with the specified name already exists, the net it belongs to must have the same name. If the net name does not match the terminal name, the function reports an error.
To associate the shape with a terminal and net, this argument is required. If you do not want to specify this argument, omit it or specify its value as `nil`.
Default: `nil`

`?termIOType` *S_termIOType*

Symbol or character string specifying the direction type (I/O type) for the terminal. If *S_termName* is not specified, the *S_termIOType* argument is ignored. If a terminal with the name specified by *S_termName* already exists, it must have the same direction type as specified by *S_termIOType*; if the direction type is not the same, the function reports an error.
Valid Values: `input`, `output`, `inputOutput`, `switch`, `jumper`
Default: `inputOutput`

Symbol or character string specifying the direction type (I/O type) for the terminal. If *S_termName* is not specified, the *S_termIOType* argument is ignored. If a terminal with the name specified by *S_termName* already exists, it must have the same direction type as specified by *S_termIOType*; if the direction type is not the same, the function reports an error.
Valid Values: `input`, `output`, `inputOutput`, `switch`, `jumper`
Default: `inputOutput`

`?pin` *g_pin*

Boolean value indicating whether or not to make the shape into a pin. The value must be `t` or `nil`. When you specify *g_pin*, you must also specify a value other than `nil` for *S_netName* and/or *S_termName*; if you do not, all connectivity arguments are ignored but no error is reported. If you do not want to specify this argument, omit it or specify its value as `nil`. When the value is `nil`, the shape created is not a pin and all other arguments containing the word `pin` are ignored.
Valid Values: `t`, `nil`
Default: `nil`

`?pinAccessDir` *tl_pinAccessDir*

Text string or list specifying the access direction(s) for the pin. Enclose string values in quotation marks. Do not use a symbol. An example of a list is
`list( "top" "bottom" )`
Valid Values: `top`, `bottom`, `left`, `right`, `any`, `none`, or a list containing any of these values
Default: `any`

`?pinLabel` *`g_pinLabel`*

Boolean value indicating whether or not to add a text display object for the pin, where the text is the contents of the *`S_termName`* argument. The value must be `t` or `nil`. When the value is `t`, the system creates a text display object as a ROD object and aligns it to the pin using the values you specify for the other pin label arguments. When the value is `nil`, no label is created and all other arguments containing the characters `pinLabel` are ignored.
Valid Values: `t`, `nil`
Default: `nil`

`?pinLabelHeight` *`n_pinLabelHeight`*

Positive integer or floating-point number specifying the vertical height of the pin text-display label in user units.
Default: `1`

`?pinLabelLayer` *`txl_pinLabelLayer`*

Text string, integer, or list specifying the layer or layer-purpose pair for the pin text-display label, such as

```
?pinLabelLayer "metal1"
?pinLabelLayer 45
?pinLabelLayer list("metal1" "drawing")
```

Enclose string values in quotation marks. Do not use a symbol. For lists, use the following format:
`list(`*`layer purpose`*`)`

Examples of layer-purpose pairs:

```
?pinLabelLayer list("metal1" "drawing")
?pinLabelLayer list(45 252)
?pinLabelLayer list(45 "drawing")
?pinLabelLayer list("metal1" 252)
```

When you specify only the layer, the purpose defaults to the purpose defined by *`l_layer`*. When you specify neither layer nor purpose, both layer and purpose default to the values defined by *`l_layer`*.
Default: layer specified by *`l_layer`*; purpose specified by *`l_layer`*

`?pinLabelFont` *`S_pinLabelFont`*

Symbol or character string specifying the name of the font for the pin text-display label.

Valid Values: `euroStyle`, `gothic`, `math`, `roman`, `script`, `stick`, `swedish`
Default: `stick`

`?pinLabelDrafting` *`g_pinLabelDrafting`*

Boolean value indicating whether or not to allow rotation of the pin text-display label by more than 90 degrees. The value must be `t` or `nil`. You can set or change label rotation with the *`S_pinLabelOrient`* argument or in the layout editor. When the value is `t`, the label can rotate less than or equal to 90 degrees. When the value is `nil`, the label can rotate more than 90 degrees.
Valid Values: `t`, `nil`
Default: `t`

`?pinLabelpinLabelOrient` *`S_pinLabelOrient`*

Symbol or character string specifying the orientation of the pin text-display label.

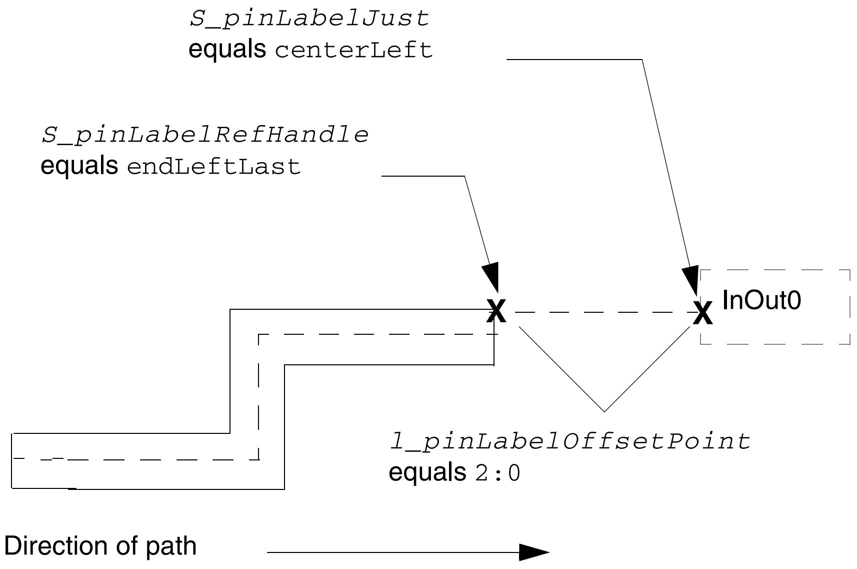
Valid Values:

0	R0
90	R90
180	R180
270	R270
MY	sideways
MYR90	sideways & 90
MX	upsideDown
MXR90....	sideways & 270

Default: `R0`

`?pinLabelOffsetPoint` *`l_pinLabelOffsetPoint`*

Lists one set of coordinates specifying the X and Y offset of the origin of the pin text-display label from a point handle on the shape. The label origin is specified by *`S_pinLabelJust`*, and the shape point handle is specified by *`S_pinLabelRefHandle`*.



Use one of the following formats:
`x:y`
or
`list(x y)`
or any expression that evaluates to a list of two numbers.
Default: `0:0`

`?pinLabelJust` *`S_pinLabelJust`*

Symbol or character string specifying the origin point for the text-display label.
Valid Values:

`upperLeft`	or	`uL`	`lowerLeft`	or	`lL`
`upperCenter`	or	`uC`	`lowerCenter`	or	`lC`
`upperRight`	or	`uR`	`lowerRight`	or	`lR`
`centerLeft`	or	`cL`			
`centerCenter`	or	`cC`			
`centerRight`	or	`cR`			

Default: `centerCenter`

`?pinLabelRefHandle` *`S_pinLabelRefHandle`*

Symbol or character string specifying the point handle on the shape from which you want to offset the origin point of the text-display label. You can specify the name of any bounding box or segment point handle.

For a detailed description of system-defined point handles, see "System-Defined Handles" in *Virtuoso Relative Object Design User Guide*.

Valid Values for bounding box point handles:

`upperLeft`	or	`uL`	`lowerLeft`	or	`lL`
`upperCenter`	or	`uC`	`lowerCenter`	or	`lC`
`upperRight`	or	`uR`	`lowerRight`	or	`lR`
`centerLeft`	or	`cL`			
`centerCenter`	or	`cC`			
`centerRight`	or	`cR`			

Valid Values for segment point handles for all ROD shapes:

`start0`	`mid0`	`end0`
`startLast`	`midLast`	`endLast`
`start`*`n`*	`mid`*`n`*	`end`*`n`*

Valid Values for segment point handles for ROD paths:

`start0`	`mid0`	`end0`
`startLast`	`midLast`	`endLast`
`start`*`n`*	`mid`*`n`*	`end`*`n`*
`startLeft0`	`midLeft0`	`endLeft0`
`startLeftLast`	`midLeftLast`	`endLeftLast`
`startLeft`*`n`*	`midLeft`*`n`*	`endLeft`*`n`*
`startRight0`	`midRight0`	`endRight0`
`startRightLast`	`midRightLast`	`endRightLast`
`startRight`*`n`*	`midRight`*`n`*	`endRight`*`n`*

Default: the bounding box point handle `centerCenter`

Offset Subpath Arguments (l_offsetSubpathArgs)

For a detailed description of offset subpaths, see "Offset Subpaths" in *Virtuoso Relative Object Design User Guide*.

`?layer` *`txl_layer`*

Text string, integer, or list specifying the layer or layer-purpose pair for the subpath, such as:

```
?layer "metal1"
?layer 45
?layer list("metal1" "drawing")
```

Enclose string values in quotation marks. Do not use a symbol. Use the one of the formats defined for the master path argument *`txl _layer`*. You are required to specify a layer.

Default Purpose: `drawing`

`?width` *`n_width`*

Positive integer or floating-point number specifying the width of the offset subpath.

If you do not specify the width, the system uses the `minWidth` rule for the offset subpath layer from the technology file. If the `minWidth` rule is not defined in the technology file, `rodCreatePath` reports an error.

Default: `minWidth` for the offset subpath layer from the technology file

`?sep` *`n_sep`*

Signed integer or floating-point number specifying the separation between the centerline or an edge of the subpath and the centerline or an edge of the master path, depending on the value of *`S_justification`*.
Default: `0`

`?justification` *`S_justification`*

Symbol or character string specifying from which part of the master path to separate the subpath, in relation to the direction of the master path. Enclose character strings in quotation marks (""); precede symbols with a single quotation mark ('). ROD converts symbols to character strings.

When `left`, the right edge of the subpath is separated from the left edge of the master path. When `right`, the left edge of the subpath is separated from the right edge of the master path. When `center`, the centerline of the subpath is separated from the centerline of the master path.
Valid Values: `center`, `left`, `right`
Default: `center`

`?beginOffset` *`n_beginOffset`*

Signed integer or floating-point number specifying the starting edge of the subpath in relation to the starting edge of the master path. A positive number extends the end of the subpath beyond the end of the master path; a negative number retracts the end of the subpath from the end of the master path.
Default: *`n_endOffset`* if specified; otherwise `0`

`?endOffset n_endOffset`

Signed integer or floating-point number specifying the ending edge of the subpath in relation to the ending edge of the master path. A positive number extends the end of the subpath beyond the end of the master path; a negative number retracts the end of the subpath from the end of the master path.
Default: *`n_beginOffset`* if specified; otherwise `0`

`?choppable` *`g_choppable`*

Boolean value indicating whether or not the subpath can be chopped. The value must be `t` or `nil`. When the master path is choppable, all subpaths must be choppable. When the master path is not choppable, each subpath can be choppable or not.
Default: `t`

`?prop` *`l_prop`*

A list or lists of lists, where each list contains the name of a property and its value. When the property name is a string, enclose it in quotes. For a description of the valid data types, see the “About the Add Property Form” in the *Virtuoso Layout Suite L User Guide*. For Boolean properties, the value must be `t` or `nil`. For example, you would specify a single Boolean property as follows:

```
?prop list("myProp" t)
```

and a list of properties as follows:

```
?prop list(
   list("myProp1" propValue1)
   list("myProp2" propValue2)
   ...
   list("myPropN" propValueN)
)
```

For a more detailed description about specifying lists of lists, see Formatting List-of-Lists Arguments for Subparts.
Default: `nil`

Enclosure Subpath Arguments (l_encSubpathArgs)

For a detailed description of enclosure subpaths, see “Enclosure Subpaths”.

`?layer` *`txl_layer`*

Text string, integer, or list specifying the layer or layer-purpose pair for the subpath, such as

```
?layer "metal1"
?layer 45
?layer list("metal1" "drawing")
```

Enclose string values in quotation marks. Do not use a symbol. Use the one of the formats defined for the master path argument *`txl_layer`*. You are required to specify a layer.
Default Purpose: `drawing`

`?enclosure` *`n_enclosure`*

Signed integer or floating-point number specifying the enclosure of the subpath in relation to the master path. The system computes the width of an enclosure subpath as follows:

width = (*`n_width`* of master path) – (2 * *`n_enclosure`*)

If you do not specify this argument, the system uses the `minEnclosure` rule from the technology file for master path layer to subpath layer; if the `minEnclosure` rule is not defined in the technology file, `rodCreatePath` reports an error.
Default: `minEnclosure` rule from the technology file for master path layer to subpath layer

`?beginOffset` *`n_beginOffset`*

Signed integer or floating-point number specifying the starting edge of the subpath in relation to the starting edge of the master path. A positive number extends the end of the subpath beyond the end of the master path; a negative number retracts the end of the subpath from the end of the master path.
Default: *`n_endOffset`* if specified; otherwise the negative of *`n_enclosure`*

`?endOffset` *`n_endOffset`*

Signed integer or floating-point number specifying the ending edge of the subpath in relation to the ending edge of the master path.

A positive number extends the end of the subpath beyond the end of the master path; a negative number retracts the end of the subpath from the end of the master path.
Default: *`n_beginOffset`* if specified; otherwise the negative of *`n_enclosure`*

`?choppable` *`g_choppable`*

Boolean value indicating whether or not the subpath can be chopped. The value must be `t` or `nil`. When the master path is choppable, all subpaths must be choppable. When the master path is not choppable, each subpath can be choppable or not.
Default: `t`

`?prop` *`l_prop`*

A list or lists of lists, where each list contains the name of a property and its value. When the property name is a string, enclose it in quotes. For a description of the valid data types, see the "About the Add Property Form" in the *Virtuoso Layout Suite L User Guide*. For Boolean properties, the value must be `t` or `nil`. For example, you would specify a single Boolean property as follows:

```
?prop list("myProp" t)
```

and a list of properties as follows:

```
?prop list(
   list("myProp1" propValue1)
   list("myProp2" propValue2)
   ...
   list("myPropN" propValueN)
)
```

For a more detailed description about specifying lists of lists, see Formatting List-of-Lists Arguments for Subparts.
Default: `nil`

Subrectangle Arguments (l_subrectArgs)

For a detailed description of sets of subrectangles for `rodCreatePath`, see "Sets of Subrectangles".

`?layer` *`txl_layer`*

Text string, integer, or list specifying the layer or layer-purpose pair for the subrectangle(s), such as

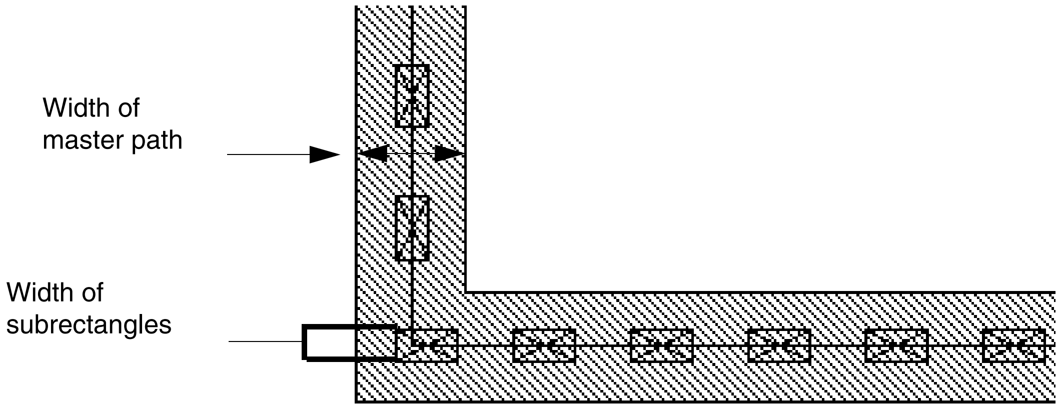
```
?layer "metal1"

?layer 45
?layer list("metal1" "drawing")
```

Enclose string values in quotation marks. Do not use a symbol. Use the one of the formats defined for the master path argument *`txl_layer`*. You are required to specify a layer.
Default Purpose: `drawing`

`?width` *`n_width`*

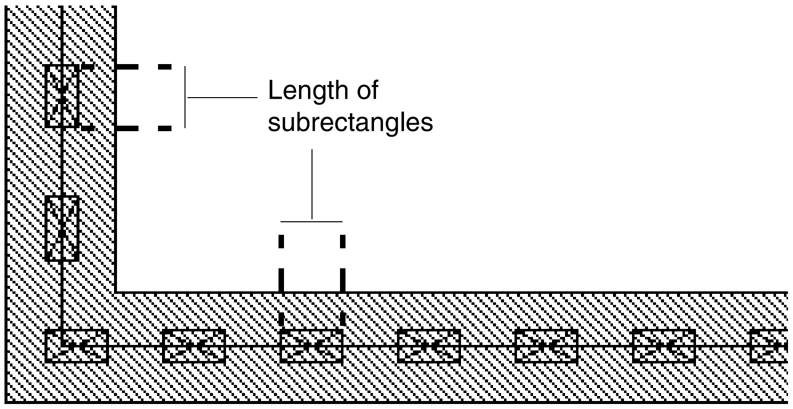
Positive integer or floating-point number specifying the width of the rectangle(s), where the width is parallel to the width of the master path.

If not specified, the system uses *`n_length`*; if neither is specified, the system uses the `minWidth` rule for the subrectangle layer from the technology file. If the `minWidth` rule is not defined in the technology file, `rodCreatePath` returns `nil` to indicate an error.
Default: *`n_length`* if specified; otherwise `minWidth` for the subrectangle layer from the technology file



`?length` *`n_length`*

Positive integer or floating-point number specifying the length of the subrectangle(s), where the length is parallel to the master path centerline.

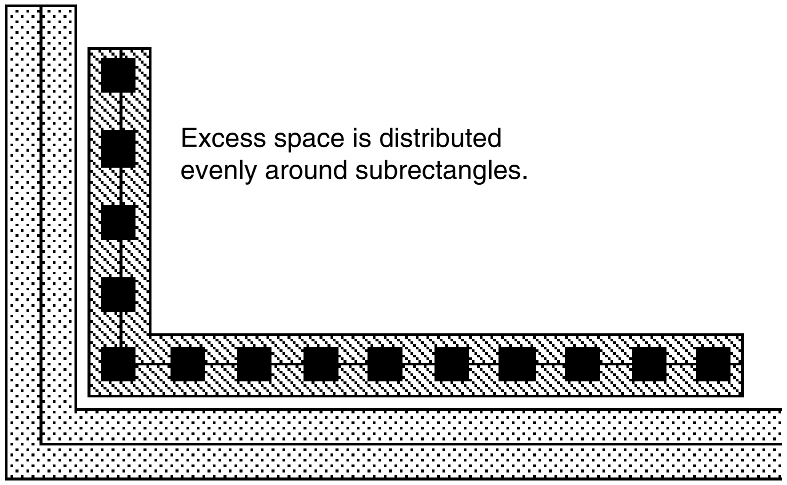


If not specified, the system uses *`n_width`*; if neither is specified, the system uses the `minWidth` rule for the subrectangle layer from the technology file. If the `minWidth` rule is not defined in the technology file, `rodCreatePath` returns `nil` to indicate an error.
Default: *`n_width`* if specified; otherwise, `minWidth` for the subrectangle layer from the technology file

`?gap` *`S_gap`*

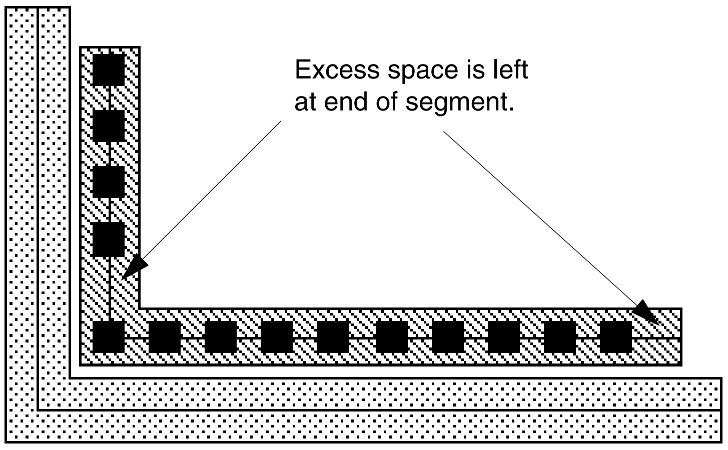
Symbol or character string specifying the method the system uses to place subrectangles within each segment. Enclose character strings in quotation marks (""); precede symbols with a single quotation mark ('). ROD converts symbols to character strings.

The system always uses the value of *`n_space`* for minimum space between subrectangles and calculates the maximum number of rectangles that fit in the segment, allowing for the space needed by *`n_beginOffset`* and *`n_endOffset`*. When the value of *`S_gap`* is `distribute`, the system distributes the space around subrectangles as evenly as possible, in multiples of the grid space specified by `mfgGridResolution`. Any remaining space is left after the last subrectangle in the segment.



When the value of *`S_gap`* is `distribute` and there is only one subrectangle, the value of the `distributeSingleSubRect` environment variable determines where the single subrectangle is placed. When `nil`, which is the default, the subrectangle is offset as specified by *`n_beginOffset`* and *`n_endOffset`*. When `t`, the subrectangle is centered (but placed on grid), and *`n_beginOffset`* and *`n_endOffset`* are ignored.

When the value of *`S_gap`* is `minimum`, the system places subrectangles *`n_space`* apart until there is no space for another rectangle, then leaves the excess space after the last subrectangle in the segment.



Valid Values: `distribute`, `minimum`
Default: `distribute`

`?sep` *`n_sep`*

Signed integer or floating-point number specifying the separation between the centerline or an edge of the subrectangles and the centerline or an edge of the master path, depending on the value of *`S_justification`*. The system places subrectangles on grid, as close to the specified separation as possible.
Default: `0`

`?justification` *`S_justification`*

Symbol or character string specifying from which part of the master path to separate the subrectangles, in relation to the direction of the master path. When `left`, the right edge of the subrectangles is separated from the left edge of the master path. When `right`, the left edge of the subrectangles is separated from the right edge of the master path. When `center`, the centerline of the subrectangles is separated from the centerline of the master path.
Valid Values: `center`, `left`, `right`
Orthogonal configuration
Default: `center`

`?beginOffset` *`n_beginOffset`*

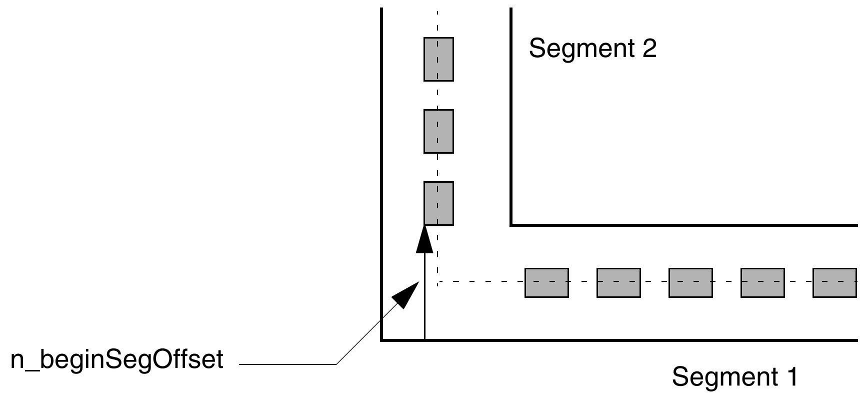
Signed integer or floating-point number specifying the offset of the edge of the first subrectangle from the starting edge of the master path. A positive number extends the beginning of the subrectangles beyond the beginning of the master path; a negative number retracts the beginning of the subrectangles from the beginning of the master path.
Default: *`n_endOffset`* if specified; otherwise `0`

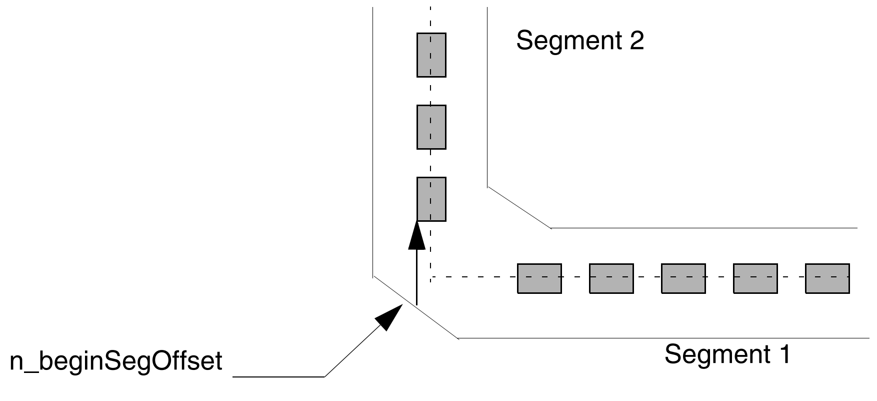
`?endOffset` *`n_endOffset`*

Signed integer or floating-point number specifying the offset of the edge of the last subrectangle from the ending edge of the master path. A positive number extends the end of the subrectangles beyond the end of the master path; a negative number retracts the end of the subrectangles from the end of the master path.
Diagonal Configuration
Default: *`n_beginOffset`* if specified; otherwise `0`

`?beginSegOffset` *`n_beginSegOffset`*

Positive, signed integer or floating-point number that, together with the other offset arguments, lets you control whether there is a subrectangle in the corner of a segment. Specifies the offset of the first subrectangle in a segment measured from the edge of the master path. Does not apply to the first segment; the position of the first subrectangle in the first segment is determined by the value of the *`n_beginOffset`* argument.

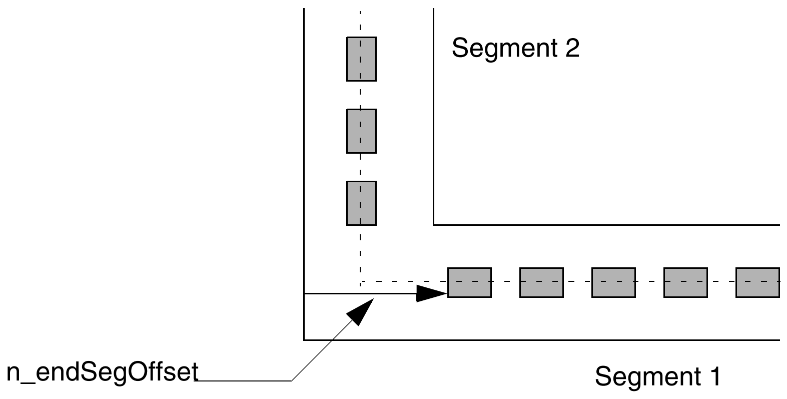




If the segment begins before the full distance specified by *`n_beginSegOffset`*, then *`n_beginSegOffset`* is not applied, and a subrectangle could be placed at the corner of the subrectangle path. When applying offset specifications, the system will not create a spacing error; if the first subrectangle in a segment would be too close to the last subrectangle in the previous segment, the system discards the last subrectangle in the previous segment. This could result in more space at the corner than desired. Default: `0.0`

`?endSegOffset` *`n_endSegOffset`*

Positive, signed integer or floating-point number that, together with the other offset arguments, lets you control whether there is a subrectangle in the corner of a segment. Specifies the offset of the last subrectangle in a segment from the edge of the master path. Does not apply to the last segment; the position of the last subrectangle in the last segment is determined by the value of the *`n_endOffset`* argument.



If the segment ends before the full distance specified by *`n_endSegOffset`*, then *`n_endSegOffset`* is not applied, and a subrectangle could be placed at the corner of the subrectangle path.
Default: `0.0`

`?space` *`n_space`*

Positive integer or floating-point number specifying the distance between the edges of adjoining rectangles. If not specified, the system uses the `minSpacing` rule for the subrectangle layer from the technology file. If the `minSpacing` rule is not defined in the technology file, `rodCreatePath` returns `nil` to indicate an error.
Default: `minSpacing` for the subrectangle layer from the technology file

`?choppable` *`g_choppable`*

Boolean value indicating whether or not the subrectangle(s) can be chopped. The value must be `t` or `nil`. When the master path is choppable, all subrectangles must be choppable. When the master path is not choppable, each set of subrectangles can be choppable or not.
Default: `t`

`?prop` *`l_prop`*

A list or lists of lists, where each list contains the name of a property and its value. When the property name is a string, enclose it in quotes. For a description of the valid data types, see the "About the Add Property Form" in the *Virtuoso Layout Suite L User Guide*. For Boolean properties, the value must be `t` or `nil`. For example, you would specify a single Boolean property as follows:

```
?prop list("myProp" t)
```

and a list of properties as follows:

```
?prop list(
   list("myProp1" propValue1)
   list("myProp2" propValue2)
   ...
   list("myPropN" propValueN)
)
```

For a more detailed description about specifying lists of lists, see Formatting List-of-Lists Arguments for Subparts.
Default: `nil`

`?diagonal` *`g_diagonalSubRect`*

Optional keyword-value pair argument specifying whether to create subrectangles on diagonal portions of the path.

When defined, this argument must appear at the end of the subrectangle argument list, after any list specifying connectivity, immediately prior to the closing parentheses. Setting this argument to `t` works only when diagonal path segments are 45 degrees and all subrectangles on the path are square. When specified, you must type the keyword, followed by `t` or `nil`. For example, type: `?diagonal t`
Default: `nil`

Value Returned

`R_rodObj`	The ROD object ID for the single layer or multipart ROD path that was created.
`nil`	No path was created due to an error.

Additional Information

Creating Self-Intersecting Paths

You cannot create self-intersecting paths using `rodCreatePath`. If you specify the function arguments so that the master path and/or one or more subpaths would self-intersect, the system will not create the path.

For example, if you create a guard ring with abutted ends, and the ends of a subpath have a positive offset, the subpath will extend beyond the master path, which could cause intersection.

If your code tries to create a self-intersecting path, the path is not created, and the system displays a warning in the CIW.

Specifying Arguments as nil

For the `rodCreatePath` function, a value of `nil` means the following, depending on the type of argument:

- For Boolean arguments, `nil` is a valid value, and the system uses it. The `rodCreatePath` Boolean arguments are:

`g_choppable`	*`g_pin`*
`g_pinLabel`	*`g_pinLabelDrafting`*

- For non-Boolean arguments, specifying `nil` causes the system to use the default as it is defined in this document for that argument. Specifying `nil` is equivalent to not specifying a value for the argument.
- For the following Connectivity arguments, the default value is `nil`:

  `S_netName` *`S_termName`* *`g_pin`*

  When you specify any of the arguments listed above as `nil`, it is equivalent to not specifying a value for the argument.

Specifying Pins

When you specify the *`g_pin`* argument, you must also specify the *`S_netName`* and/or *`S_termName`* argument with a value other than `nil`. If you specify *`g_pin`* but do not specify *`S_netName`* and/or *`S_termName`* with a value other than `nil`, the system ignores all connectivity arguments but does not report an error.

Formatting List-of-Lists Arguments for Subparts

You must use a **list of lists** to specify arguments for an offset subpath, enclosure subpath, or set of subrectangles. The top-level list contains one or more sublists. Each sublist defines a unique subpath or set of subrectangles.

Each sublist consists of one or more keyword-value pairs, where the keyword identifies the argument. To specify the value as a character string, enclose it in quotation marks (""); to specify the value as a symbol, precede it with a single quotation mark (').

For any keyword that returns either a string or symbol, you can specify a Cadence SKILL language expression (constant, variable, or function call) as the value.

As an example, the syntax format for specifying an enclosure subpath as part of a `rodCreatePath` function is shown on the following page.

```
?encSubPath  list( list( ?layer         txl_layer
                         ?enclosure     n_enclosure
                         ?beginOffset   n_beginOffset
                         ?endOffset     n_endOffset
                         ?choppable     g_choppable
                       ) ;End of first enclosure subpath

                   list( ?layer         txl_layer
                         ?enclosure     n_enclosure
                         ?beginOffset   n_beginOffset
                         ?endOffset     n_endOffset
                         ?choppable     g_choppable
                       );End of second enclosure subpath
                                 .
```

```
                              .
                              .
            ) ;End of encSubPath list
```

An example showing values for an enclosure subpath within a `rodCreatePath` function specification might look like this:

```
poly1EndOffset = -0.4
tfId = = techGetTechFile( pcCellView )
?encSubPath list(    list( ?layer         "metal2"
                           ?enclosure
                                 techGetParam( tfId "m1m2enc" )
                           ?beginOffset   -.4
                           ?choppable     nil
                         ) ; end of 1st enclosure subpath
                     list( ?layer         "poly1"
                           ?enclosure     .1
                           ?beginOffset   0
                           ?endOffset     poly1EndOffset
                           ?choppable     nil
                         ) ; end 2nd enclosure subpath
            ) ; end of 2nd enclosure subpath
```

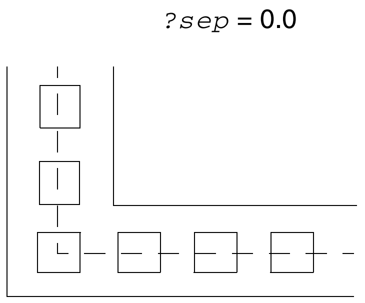
where `poly1EndOffset` is a variable and `techGetParam( tfId "m1m2enc" )` is a function call.

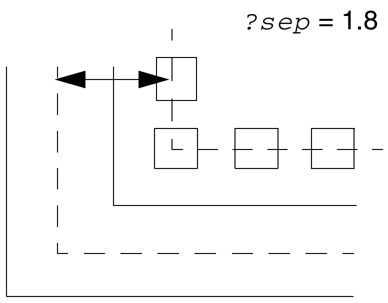
How the System Follows to Create Subrectangles

The system follows these steps to create subrectangles:

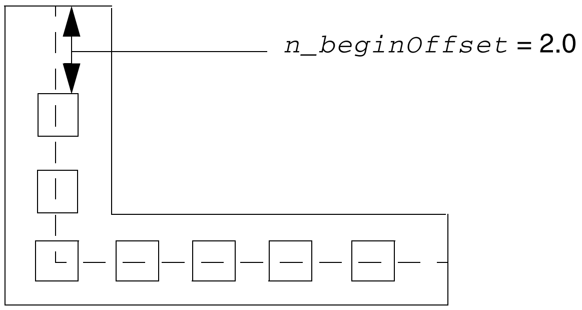
1. Creates all subrectangles on the specified layer, with the subrectangle centerline separated from the master path centerline by *`n_sep`*.

   When *`n_sep`* is zero, the master path centerline bisects the width of the subrectangles. When *`n_sep`* is less than or greater than zero, the subrectangles are offset from the master path centerline.



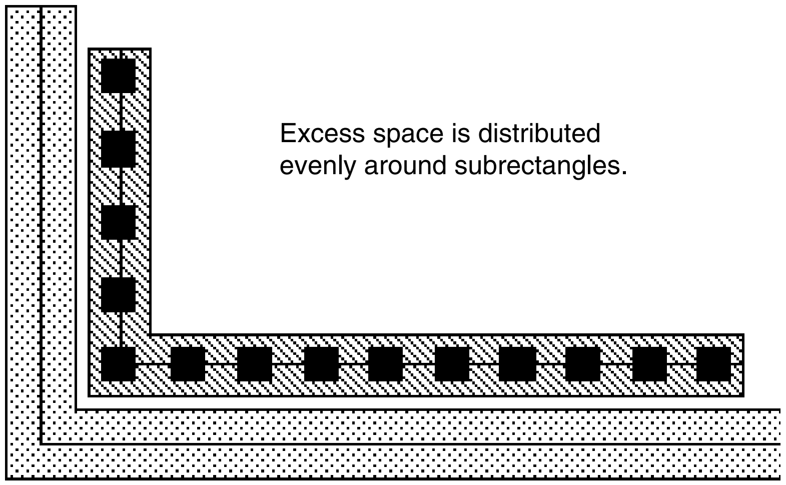


2. For the first path segment, offsets the edge of the first subrectangle from the starting edge of the master path by the distance specified in *`n_beginOffset`*.

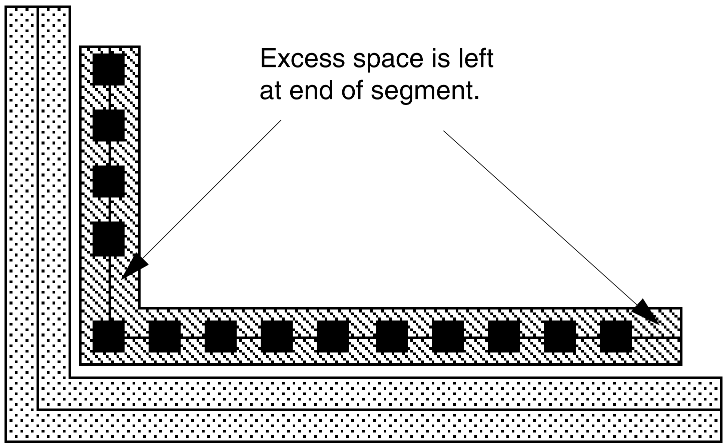


3. Uses the value of *`n_space`* to compute the maximum number of subrectangles that fit on grid in the segment, and the value of *`S_gap`* to determine what to do with excess space, as follows:

   ❑ When *`S_gap`* equals `distribute` (the default), distributes the space as evenly as possible, in multiples of the grid space specified by `mfgGridResolution`.

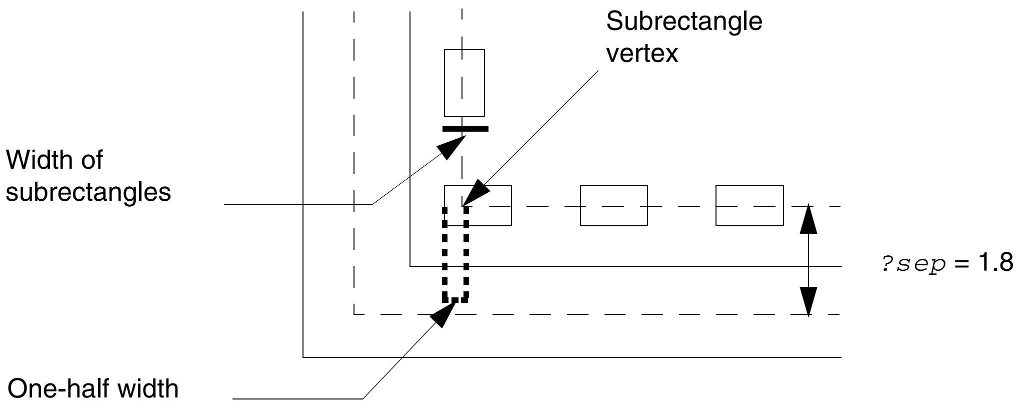


- ❑ When *S_gap* equals `minimum`, places subrectangles *n_space* apart until there is no space for another rectangle, then leaves the excess space after the last subrectangle in the segment.



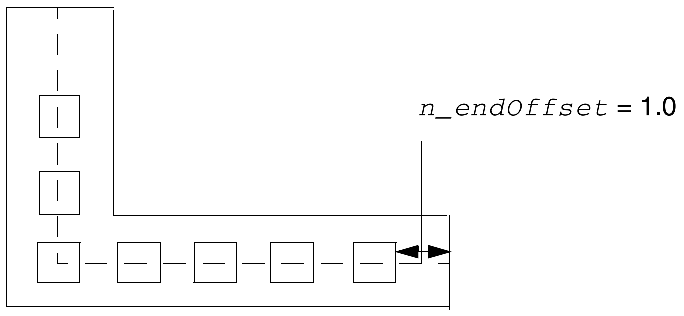
4. For each subsequent path segment, the system

- ❑ Positions the first subrectangle on the vertex of the subrectangle centerline with half its width towards the beginning of the segment.



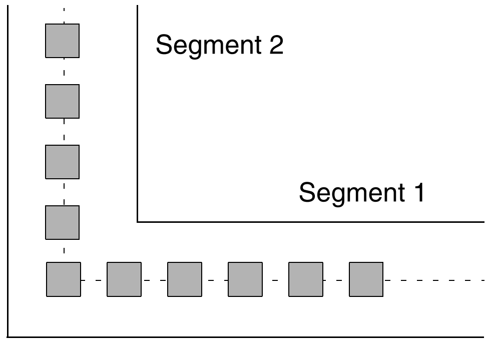
- ❑ Repeats step 3.

5. For the last path segment, the system offsets the edge of the last subrectangle from the ending edge of the master path by the distance specified in *`n_endOffset`*. For the remaining subrectangles in the last segment, the system repeats step 3.



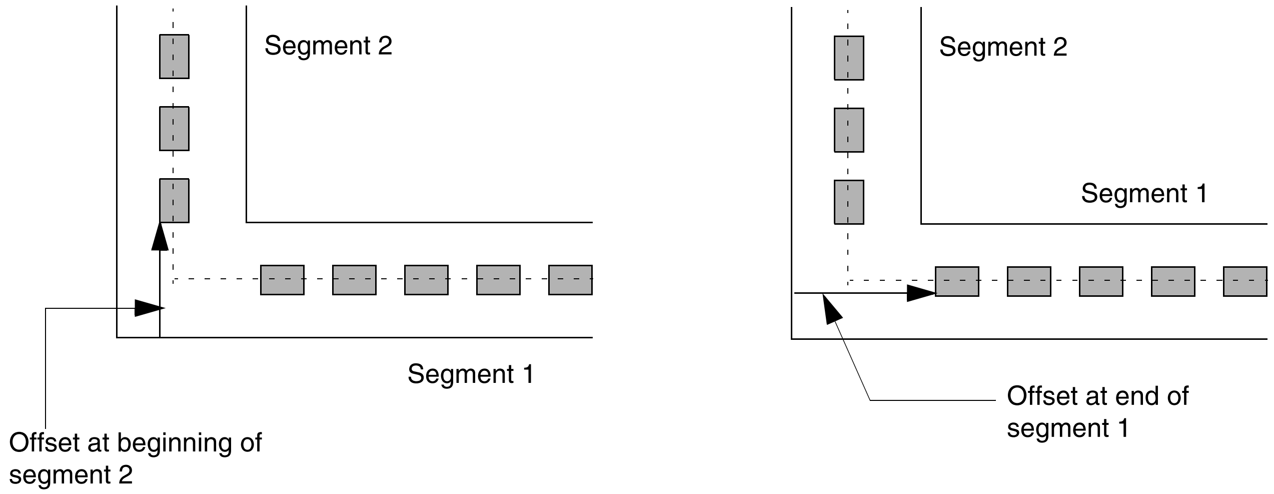
Subrectangles in the Corners of Segments

By default, the system creates a subrectangle in the corner formed by each segment of a subrectangle subpath, and then fills in the rest of the segment with subrectangles, following the spacing you specify with the `rodCreatePath` function.



There are several reasons that you might want to keep subrectangles out of the corners of your subrectangle subpaths. For example, when you specify multiple subrectangle subpaths, placing rectangles in the corners of segments might cause individual subrectangles to be out of alignment vertically or horizontally with the subrectangles in the other subrectangle subpaths.

If you do not want subrectangles in the corners of segments, you can keep them out by specifying the *`n_beginSegOffset`* and *`n_endSegOffset`* arguments. The system measures the *`n_beginSegOffset`* and *`n_endSegOffset`* values from the edge of the master path, as shown below.



When specifying the segment offset arguments, keep in mind:

- The *`n_beginSegOffset`* value does not apply to the first segment; the position of the first subrectangle in the first segment is determined by the value of the *`n_beginOffset`* argument.
- The *`n_endSegOffset`* value does not apply to the last segment; the position of the last subrectangle in the last segment is determined by the value of the *`n_endOffset`* argument.
- The segment offset arguments work in combination with the *`n_beginOffset`* and *`n_endOffset`* arguments.
- If the segment begins before the full distance specified by the *`n_beginSegOffset`* argument, then *`n_beginSegOffset`* is not applied, and a subrectangle could be placed at the corner of the subrectangle subpath.
- If the segment ends before the full distance specified by the *`n_endOffset`* argument, then *`n_endOffset`* is not applied, and a subrectangle could be placed at the corner of the subrectangle subpath.
- When applying the offset options, the system will not create a spacing error. If the first subrectangle in a subrectangle subpath segment would be too close to the last subrectangle in the previous segment, the system discards the last subrectangle in the previous segment.

When you specify the offset options, you need to check the results to make sure the corners are as desired. If they are not, you should change the values of the offset arguments and try again.

Disconnecting Shapes in a Multipart Path

You can change a multipart path into separate, unrelated shapes by removing the name of the multipart path. Once you do this, all relationships between the shapes are removed.

➤ To change all of the shapes in a multipart path into separate, unrelated shapes, remove the name of the multipart path using the `rodUnNameShape` function.

Now the master path is a separate, unnamed path; each subpath is a separate, unnamed path; and each rectangle is a separate, unnamed shape. None of them is a ROD object, and there is no relationship between them.

Making a Former MPP Part into a ROD Object

You can make any shape that was formerly part of a multipart path (the master path, any subpath, or any individual subrectangle) into a ROD object by assigning it a unique name.

➤ To make a shape into a ROD object, assign a unique name to the shape using the `rodNameShape` function.

Each shape to which you assign a name is now a separate, unique ROD object.

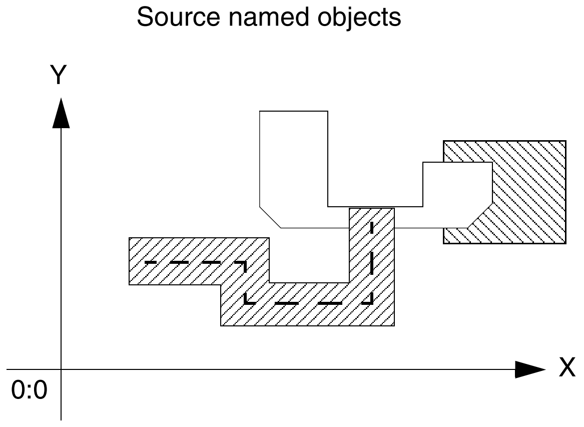
rodCreatePolygon

```
rodCreatePolygon(
     [ ?name S_name ]
     [ ?layer txl_layer ]
     [ ?pts l_pts ]
     [ ?cvId d_cvId ]
     [ ?fromObj Rl_fromObj ]
     [ ?size txf_size ]
     [ ?prop l_prop ]
     ; ROD Connectivity Arguments for Polygons
     [ ?netName S_netName ]
     [ ?termName S_termName ]
     [ ?termIOType S_termIOType ]
     [ ?pin g_pin ]
     [ ?pinAccessDir tl_pinAccessDir ]
     [ ?pinLabel g_pinLabel ]
     [ ?pinLabelHeight n_pinLabelHeight ]
     [ ?pinLabelLayer txl_pinLabelLayer ]
     [ ?pinLabelFont S_pinLabelFont ]
     [ ?pinLabelDrafting g_pinLabelDrafting ]
     [ ?pinLabelOrient S_pinLabelOrient ]
     [ ?pinLabelOffsetPoint l_pinLabelOffsetPoint ]
     [ ?pinLabelJust S_pinLabelJust ]
     [ ?pinLabelRefHandle S_pinLabelRefHandle ]
     );end rodCreatePolygon
     => R_rodObj / nil
```

Description

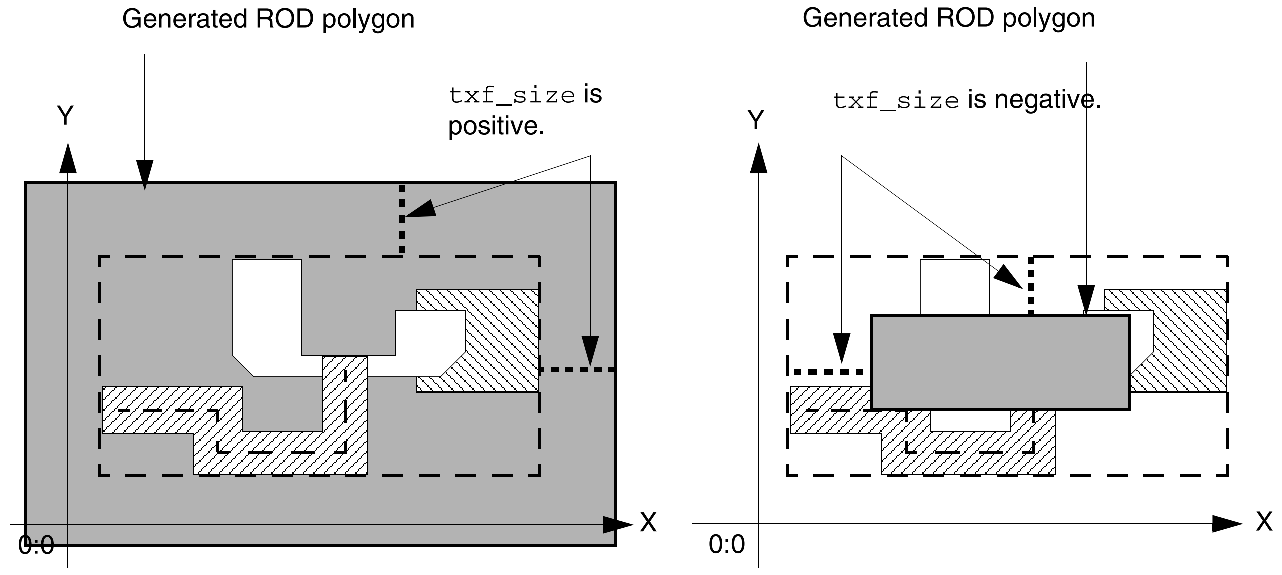
Creates one polygon from a list of points or from one or more named objects. Also creates a ROD object containing information associated with the polygon, including its name and database ID. The associated ROD object is identified by a ROD object ID. The polygon is created at level zero in the hierarchy. You can assign a property name and value, or a list of property names and values, to the polygon. You can specify connectivity for the polygon by associating it with a specific terminal and net. You can also make the polygon into a pin.

When you specify two or more existing named objects as the source for creating a polygon, as shown in the example below,



the system creates a four-sided, rectangular polygon based on a bounding box around all of the object(s).

The new rectangular polygon is larger or smaller than the bounding box, depending on whether you specify a positive or negative size.



Note: In the current release, no relationship exists between a generated polygon and its source object(s) after the new polygon is created.

For a detailed overview of creating polygons from other objects, see “Creating a Polygon from Another Object”.

Arguments

`?name` *`S_name`*

Symbol or character string specifying the name for the polygon. Enclose character strings in quotation marks (""); precede symbols with a single quotation mark (‘). ROD converts symbols to character strings.

The name must be unique for the cellview. When you do not specify a name, the system assigns a unique name, consisting of the prefix `polygon`, followed by a number. For example, for the first unnamed polygon in a cellview, the system assigns the name `polygon0`, if unique; for the second, `polygon1`, and so on.
Default: `polygon`*`n`*

`?layer` *`txl_layer`*

Text string, integer, or list specifying the layer or layer-purpose pair for the polygon, such as

```
?layer "metal1"
?layer 45
?layer list("metal1" "drawing")
```

Enclose string values in quotation marks. Do not use a symbol. For lists, use the following format:
`list(` *`layer purpose`* `)`

Examples of layer-purpose pairs:

```
?layer list("metal1" "drawing")
?layer list(45 252)
?layer list(45 "drawing")
?layer list("metal1" 252)
```

You are required to specify a layer.
Default Purpose: `drawing`

`?pts` *`l_pts`*

List of points defining the edges of the polygon. Specify a point for the beginning and end of the first edge and the end of subsequent edges. The system creates a closed shape by connecting the first and last points in the list. You cannot create a polygon with self-intersecting edges.

The system discards duplicate points and excess collinear points without considering them errors. However, you must specify at least three noncoincident, noncollinear points. Coincident points have the same coordinate values. Collinear points are on the same line. If you specify more than two points on the same line, the system uses only the first and last points specified. An example of excess collinear points is `0:10`, `0:20`, and `0:30`, where the system discards `0:20`.Use one of the following formats:
`list( x:y x:y ... )`
or
`list( list(x y) list(x y) ... )`

You must specify either *`l_pts`* or *`Rl_fromObj`*. If you specify both, the system ignores the *`l_pts`* argument.
Default: none

`?cvId` *`d_cvId`*

Database ID for the cellview in which you are creating a ROD path. If `rodCreatePath` occurs in the body of a `pcDefinePCell` function or `tcCreateCDSDeviceClass` function call, the default value is `pcCellView` or `tcCellView`, respectively; otherwise, this argument is required.
Default: none

`?fromObj` *`Rl_fromObj`*

ROD object ID or list of ROD object IDs identifying a named object or list of named objects that you want to use as a source for creating a new ROD polygon. The source objects can be instances, rectangles, polygons, paths, lines, dots, labels, and/or text-display objects. You must specify either the *`Rl_fromObj`* argument or the *`l_pts`* argument. If you specify both, the system ignores the *`l_pts`* argument.
Default: none

`?size` *`txf_size`*

A signed integer, floating-point number, or Cadence SKILL language expression specifying the difference between the size of the source object (*R1_fromObj*) and the size of the generated polygon. When the source is more than one object, *txf_size* specifies the difference between a bounding box around all of the objects and the size of the generated polygon.

A positive number creates a polygon that is larger than the source object(s); a negative number creates a polygon that is smaller than the source object(s). When you specify a SKILL expression, it must evaluate to a positive or negative integer or floating-point number. SKILL expressions are evaluated only when the new polygon is created.

If you specify a source object (*R1_fromObj*) but do not specify *txf_size*, then *txf_size* defaults to `0.0`. If you specify *txf_size*, you must also specify *R1_fromObj*, or the system issues a warning message and the function fails.
Default: `0.0`

`?prop` *`l_prop`*

A list or lists of lists, where each list contains the name of a property and its value. When the property name is a string, enclose it in quotes. For a description of the valid data types, see the "About the Add Property Form" in the *Virtuoso Layout Suite L User Guide*. For Boolean properties, the value must be `t` or `nil`. For example, you would specify a single Boolean property as follows:

```
?prop list("myProp" t)
```

and a list of properties as follows:

```
?prop list(
   list("myProp1" propValue1)
   list("myProp2" propValue2)
   ...
   list("myPropN" propValueN)
)
```

For a more detailed description about specifying lists of lists, see Formatting List-of-Lists Arguments for Subparts.
Default: `nil`

ROD Connectivity Arguments for Polygons

The connectivity arguments for polygons are the same as the connectivity arguments for rectangles. See ROD Connectivity Arguments for Rectangles (rodRectConnectivityArgs).

Value Returned

`R_rodObj`	The ROD object ID for the polygon that was created.
`nil`	No polygon was created due to an error.

rodCreateRect

```
rodCreateRect(
     [ ?name S_name ]
     [ ?layer txl_layer ]
     [ ?width n_width ]
     [ ?length n_length ]
     [ ?origin l_origin ]
     [ ?bBox l_bBox ]
     [ ?elementsX x_elementsX ]
     [ ?elementsY x_elementsY ]
     [ ?spaceX n_spaceX ]
     [ ?spaceY n_spaceY ]
     [ ?cvId d_cvId ]
     [ ?fillBBox l_fillBBox ]
     [ ?fromObj Rl_fromObj ]
     [ ?size txf_size ]
     [ ?prop l_prop ]
     [ ?returnBoolean g_returnBoolean ]

     [rodRectConnectivityArgs... ]
          ; start ROD Connectivity Arguments for Rectangles (rodRectConnectivityArgs)
          [ ?netName S_netName ]
          [ ?termName S_termName ]
          [ ?termIOType S_termIOType ]
          [ ?pin g_pin ]
          [ ?pinAccessDir tl_pinAccessDir ]
          [ ?pinLabel g_pinLabel ]
          [ ?pinLabelHeight n_pinLabelHeight ]
          [ ?pinLabelLayer txl_pinLabelLayer ]
          [ ?pinLabelFont S_pinLabelFont ]
          [ ?pinLabelDrafting g_pinLabelDrafting ]
          [ ?pinLabelOrient S_pinLabelOrient ]
          [ ?pinLabelOffsetPoint l_pinLabelOffsetPoint ]
          [ ?pinLabelJust S_pinLabelJust ]
          [ ?pinLabelRefHandle S_pinLabelRefHandle ]
          ;end ROD Connectivity Arguments for Rectangles

     [ ?subRectArray     l_subrectArgs... ]
     ;start l_subrectArgs Subrectangle Arguments (l_subrectArgs)
     list(
          list(
               [ ?layer txl_layer  ]
               [ ?width n_width ]
               [ ?length n_length ]
               [ ?gap S_gap ]
               [ ?lowerLeftOffsetX n_lowerLeftOffsetX ]
               [ ?lowerLeftOffsetY n_lowerLeftOffsetY ]
               [ ?upperRightOffsetX n_upperRightOffsetX ]
               [ ?upperRightOffsetY n_upperRightOffsetY ]
               [ ?spaceX n_spaceX ]
```

```
            [ ?spaceY n_spaceY ]
            [ ?prop l_prop ]
        ;Repeat ROD Connectivity Arguments here
        );end first subrectangle list
        ...
        ) ;end all subrectangle lists
    ;end l_subrectArgs
    );end rodCreateRect

    => R_rodObj | t / nil
```

Description

Creates a single named rectangle, one or more rows and/or columns of named rectangles, or fills a bounding box with named rectangles, where each rectangle has ROD attributes. You can create these named rectangles with the arguments *`S_name`* through *`g_returnBoolean`*. Each named rectangle is a separate object, created at level zero in the hierarchy. You can also create multipart rectangles by specifying one or more arrays of unnamed *subrectangles* for each named rectangle, where each unnamed subrectangle is an ordinary database shape, with no ROD attributes, created at level zero in the hierarchy. The named rectangles in a multipart rectangle are referred to as *master rectangles*.

You can assign a property name and value (or a list of property names and values) to named rectangles and/or to any set of unnamed subrectangles. You can specify connectivity to associate named rectangles with the same terminal and net and to turn named rectangles into pins on a specified terminal and net. You can also specify connectivity for each set of unnamed subrectangles to associate it with a terminal and net and turn each subrectangle into a pin.

Although a set of unnamed subrectangles is treated as a single shape, you can get a list of the database IDs for the individual subrectangles in the set by using the ROD object ID for the multipart path with the database access operator (`~>`) and the attribute name `subShapes`. For more information, see “Accessing ROD Object Attributes”.

Note: When you want to create a single unnamed rectangle that is a regular database shape and has no ROD attributes, use the `dbCreateRect` function, documented in the *Cadence Design Framework II SKILL Functions Reference*. When you want to create a one- or two-dimensional array of unnamed rectangles that have no ROD attributes, use the rodFillBBoxWithRects function. Regular database shapes require less overhead, resulting in faster performance.

Arguments

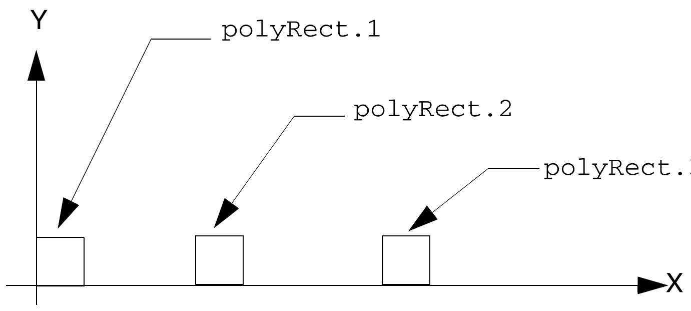
`?name` *`S_name`*

Symbol or character string specifying the name for the rectangle you want to create. Enclose character strings in quotation marks (""); precede symbols with a single quotation mark ('). ROD converts symbols to character strings.

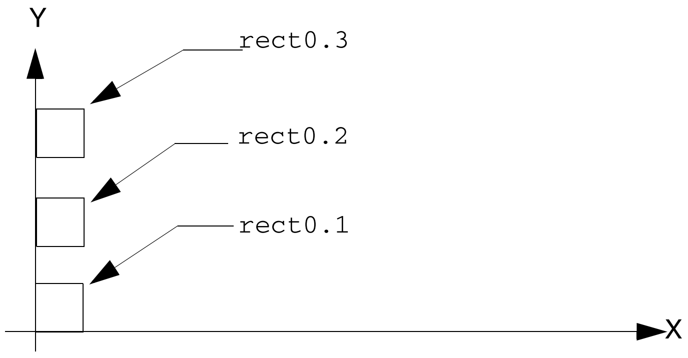
The name must be unique for the cellview. If you do not assign a name, the system assigns a unique name, consisting of the prefix `rect`, followed by a number. For example, for the first unnamed rectangle in the cellview, the system assigns the name `rect0`; for the second `rect1`, and so on.

Single row or column of rectangles: When you create a single row or column of rectangles, the system uses your specified name (or `rect`*`n`* if you did not specify a name) as a base and adds the suffix .*`n`*, where *`n`* starts at 1 and is increased by 1 for each rectangle in the row or column.

For example, if you create a row of three rectangles (rectangles in the direction of the X axis) and assign the name `polyRect`, the system names the three rectangles `polyRect.1`, `polyRect.2`, and `polyRect.3`.

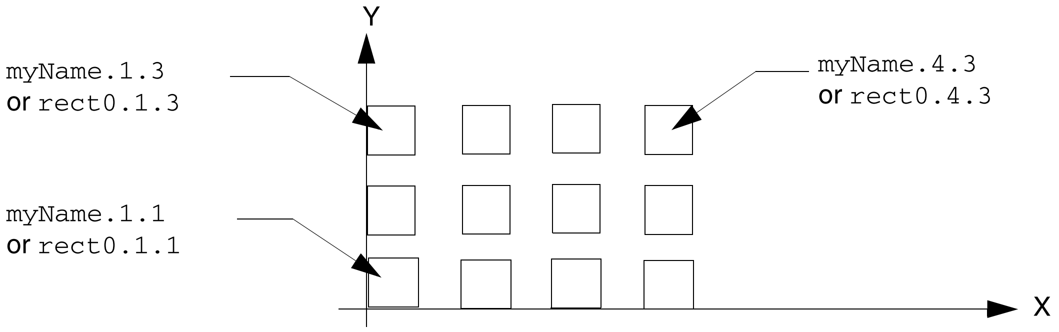


If you create a column of three rectangles (rectangles in the direction of the Y axis) and you do not assign a name, the system names the three rectangles `rect0.1`, `rect0.2`, and `rect0.3`.



Multiple rows and columns of rectangles: When you create multiple rows and columns of rectangles, the system uses your specified name as a base and adds the suffix `.`*`x`*`.`*`y`*, where `.`*`x`*`.`*`y`* is the number of the row and column.

For example, if you specify three rows and four columns, the system names the rectangles as follows:



Default: `rect`*`n`*, `rect`*`n`*`.`*`n`*, `rect`*`n`*`.`*`x`*`.`*`y`*

`?layer` *`txl_layer`*

Text string, integer, or list specifying the layer or layer-purpose pair for the rectangle(s), such as

```
?layer "metal1"
?layer 45
?layer list("metal1" "drawing")
```

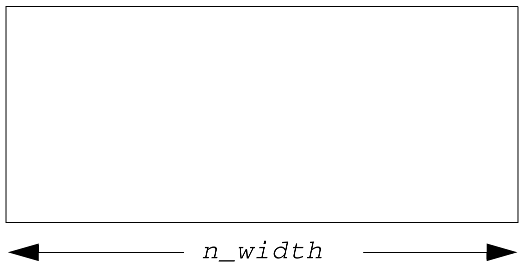
Enclose string values in quotation marks. Do not use a symbol. For lists, use the following format:

`list(` *`layer purpose`* `)`

Examples of layer-purpose pairs are
```
?layer list("metal1" "drawing")
?layer list(45 252)
?layer list(45 "drawing")
?layer list("metal1" 252)
```
You are required to specify a layer.
Default Purpose: `drawing`

`?width` *`n_width`*

Positive integer or floating-point number specifying the horizontal measurement of the rectangle, in user units.

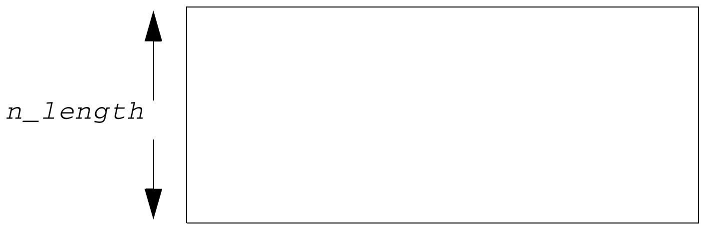


The value of *`l_bBox`* overrides the value of *`n_width`* when you specify both. If you specify neither *`n_width`* nor *`l_bBox`*, the system uses the value of *`n_length`*.

If you do not specify *`n_length`*, the system uses the value for `minWidth`, the minimum width rule for the specified layer, from your technology file. If you specify none of these and `minWidth` is not available, the function reports an error.
Default: *`n_length`* if specified; otherwise the minimum width rule for the specified layer from your technology file

`?length` *`n_length`*

Positive integer or floating-point number specifying the vertical measurement of the rectangle, in user units.



The value of *`l_bBox`* overrides the value of *`n_length`* when you specify both. If you specify neither *`n_length`* nor *`l_bBox`*, the system uses the value of *`n_width`*. If you do not specify *`n_width`*, the system uses the value for `minWidth`, the minimum width rule for the specified layer, from your technology file. If you specify none of these and `minWidth` is not available, the function reports an error.
Default: *`n_width`* if specified; otherwise the minimum width rule for the specified layer from your technology file

`?origin` *`l_origin`*

Single point or list of coordinates specifying the lower left corner of the first rectangle; use one of the following formats:
`x``:`*`y`*
or
`list(`*`x y`*`)`

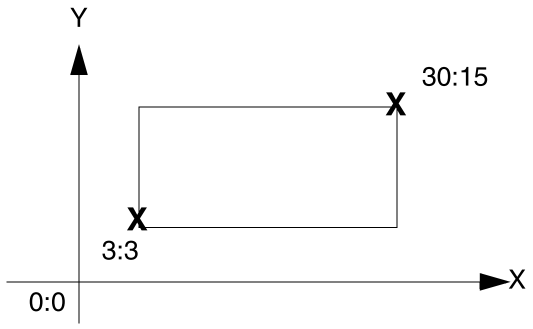
The value of *`l_bBox`* overrides the value of *`l_origin`* when you specify both.
Default: `0:0`

`?bBox` *`l_bBox`*

List of two points specifying opposite corners of a bounding box for the size of the rectangle. You can start the box in any corner. The system determines the location of the first rectangle by using the difference between the coordinates for the specified opposite corners. Use one of the following formats:
`list(` *`x`*`:`*`y x`*`:`*`y`* `)`
or
`list( list(`*`x y`*`) list(`*`x y`*`))`

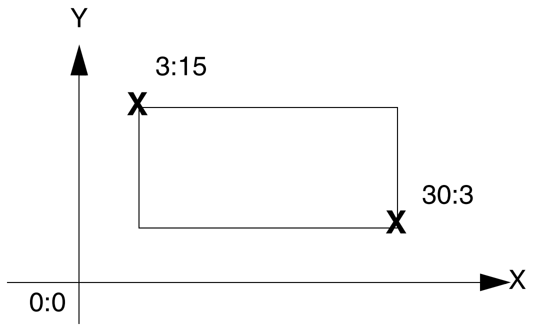
Example 1 starts in the bounding box in the lower-left corner.
`list(3:3 30:15)`
or
`list( list(3 3) list(30 15))`

or
`list( list(3 3) list(30 15))`



Example 2 starts the bounding box in the upper-left corner.

```
list(3:15 30:3)
```
or
```
list( list(3 15) list(30 3))
```



When you specify a rectangle size with *`l_bBox`*, the value of *`l_bBox`* overrides the values of *`n_width`*, *`n_length`*, and *`l_origin`*.
Default: none

`?elementsX` *`x_elementsX`*

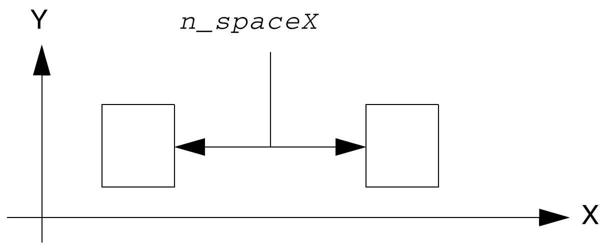
Positive integer specifying the number of rectangles to create in a row parallel to the X axis. If you also specify *`l_fillBBox`*, *`x_elementsX`* specifies the maximum number of rectangles to create in the direction of the X axis.
Default: `1`

`?elementsY` *`x_elementsY`*

Positive integer specifying the number of rectangles to create in a column parallel to the Y axis. If you also specify `l_fillBBox`, `x_elementsY` specifies the maximum number of rectangles to create in the direction of the Y axis.
Default: `1`

`?spaceX` *`n_spaceX`*

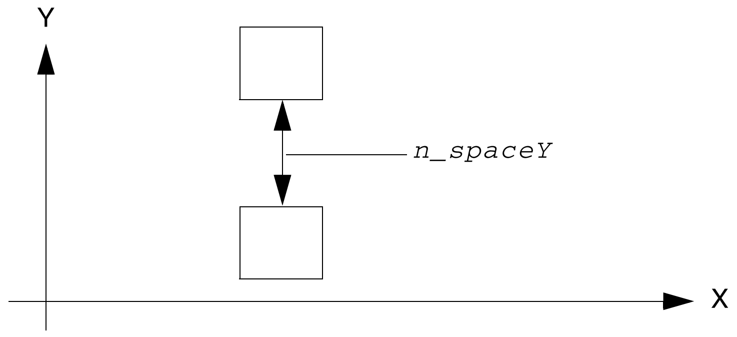
Positive integer or floating-point number specifying the distance between the edges of rectangles in the direction of the X axis. This argument is ignored when a row contains only one rectangle.



If you do not specify *`n_spaceX`*, the system uses the value of *`n_spaceY`*. If you specify neither, the system uses the value for `minSpacing`, the minimum spacing rule for the specified layer, from your technology file. If you specified neither of these and `minSpacing` is not available, the function reports an error.
Default: *`n_spaceY`* if specified; otherwise the minimum spacing rule for the specified layer from your technology file

`?spaceY` *`n_spaceY`*

Positive integer or floating-point number specifying the distance between the edges of rectangles in the direction of the Y axis. This argument is ignored when a row contains only one rectangle.



If you do not specify *n_spaceY*, the system uses the value of *n_spaceX*. If you specify neither, the system uses the value of `minSpacing`, the minimum spacing rule for the specified layer, from your technology file. If you specified neither of these and `minSpacing` is not available, the function reports an error.
Default: *n_spaceX* if specified; otherwise the minimum spacing rule for the specified layer from your technology file

`?cvId` *`d_cvId`*

Database ID for the cellview in which you are creating a rectangle.
Default: `pcCellView` or `tcCellView` when a `rodCreateRect` statement occurs in the body of a `pcDefinePCell` function or `tcCreateCDSDeviceClass` function call, respectively; `nil` if no cellview ID is specified or found

`?fillBBox` *`l_fillBBox`*

List of two points defining opposite corners of a bounding box to fill with rectangles, referred to as a *fill-bounding box*. The first point specifies any corner and the second point specifies the opposite corner. Use one of the following formats:
`list(` *`x`*`:`*`y`* *`x`*`:`*`y`* `)`
or
`list( list(`*`x y`*`) list(`*`x y`*`))`

You can specify the maximum number of rectangles the system creates in the fill-bounding box with *x_elementsX* and *x_elementsY*. If the number specified exceeds the space available in the fill-bounding box, the system creates only the number of rectangles that fit.

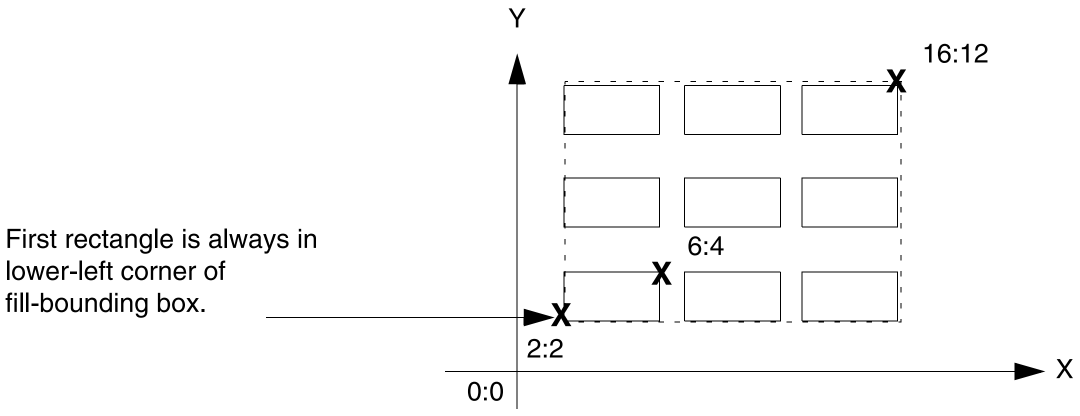
You can specify the distance between rectangles in the fill-bounding box with *`n_spaceX`* and *`n_spaceY`*.

When you use *`l_bBox`* argument with *`l_fillBBox`*, the system uses the *`l_bBox`* coordinates *only to compute the size of the rectangles*. The system always places the first rectangle in the lower-left corner of the fill-bounding box. For example, the following argument values define a 14-by-10 fill-bounding box with its lower-left corner at 2:2:

```
?llBBox list( 2:2 16:12 )
?bBox list( 0:0 4:2 )
?spaceX 1.0
?spaceY 2.0
```

The coordinates of the first rectangle (`0:0 4:2`) are used to compute the size of the rectangles, not the location of the first rectangle.

The size of the rectangles is 4 units wide by 2 units long. The rectangles are spaced 1 unit apart in the direction of the X axis and 2 units apart in the direction of the Y axis.


Y
16:12
First rectangle is always in lower-left corner of fill-bounding box.
6:4
2:2
0:0
X

The fill-bounding box is shown by a dashed line because no bounding box is created.
Default: `nil`

`?fromObj R1_fromObj`

ROD object ID or list of ROD object IDs identifying a named object or list of named objects that you want to use as a source for creating a new named rectangle. The source objects can be instances, rectangles, polygons, paths, lines, dots, labels, and/or text-display objects. When you specify `R1_fromObj`, the value of the `R1_fromObj` argument overrides other arguments that directly or indirectly specified coordinates for the generated rectangle: `n_width`, `n_length`, `l_origin`, and `l_bBox`.
Default: none

`?size txf_size`

A signed integer, floating-point number, or Cadence® SKILL language expression specifying the difference between the size of the source object (`R1_fromObj`) and the size of the generated rectangle. When the source is more than one object, `txf_size` specifies the difference between a bounding box around all of the objects and the size of the generated rectangle.

A positive number creates a rectangle that is larger than the source objects; a negative number creates a rectangle that is smaller than the source objects. When you specify a SKILL expression, it must evaluate to a positive or negative integer or floating-point number. SKILL expressions are evaluated only when the new rectangle is created.

If you specify a source object (`R1_fromObj`) but do not specify `txf_size`, then `txf_size` defaults to `0.0`. If you specify `txf_size`, you must also specify `R1_fromObj`, or the system issues a warning message and the function fails.
Default: `0.0`

`?prop l_prop`

A list or lists of lists, where each list contains the name of a property and its value. When the property name is a string, enclose it in quotes. For a description of the valid data types, see the “About the Add Property Form” in the *Virtuoso Layout Suite L User Guide*. For Boolean properties, the value must be `t` or `nil`. For example, you would specify a single Boolean property as follows:

```
?prop list("myProp" t)
```

And, a list of properties as follows:

```
?prop list(
   list("myProp1" propValue1)
   list("myProp2" propValue2)
   ...
   list("myPropN" propValueN)
)
```

For a more detailed description about specifying lists of lists, see Formatting List-of-Lists Arguments for Subparts.
Default: `nil`

`?returnBoolean` *`g_returnBoolean`*

Boolean value indicating whether or not the `rodCreateRect` function returns a Boolean value or ROD object IDs. The value must be `t` or `nil`. When *`g_returnBoolean`* is `t`, the function returns `t` or `nil`. When *`g_returnBoolean`* is `nil`, the function returns a single ROD object ID or a list of ROD object IDs. When you are creating multiple rectangles, the `rodCreateRect` function executes much more quickly if *`g_returnBoolean`* is set to `t`, so when you do not need a list of ROD object IDs, set this argument to `t`.
Default: `nil`

`rodRectConnectivityArgs...`

A list containing rod connectivity arguments for rectangles.

For more information on *`rod_RectConnectivityArgs`*, see ROD Connectivity Arguments for Rectangles (rodRectConnectivityArgs)

`?subRectArray` *`l_subrectArgs`*...

A list containing one or more lists, each of which specifies the arguments for one set of unnamed subrectangles. Each list creates a set of unnamed subrectangles for each master rectangle in the multipart rectangle; each set of subrectangles is created in relationship to the lower-left corner of each master rectangle. For information about how to specify lists of lists, see Formatting List-of-Lists Arguments for Subparts.

For more information onf *`l_subrectArgs`*, see Subrectangle Arguments (l_subrectArgs)

ROD Connectivity Arguments for Rectangles (rodRectConnectivityArgs)

`?netName` *`S_netName`*

Symbol or character string specifying the name of the net with which you want to associate the shape. Enclose character strings in quotation marks (""); precede symbols with a single quotation mark ('). ROD converts symbols to character strings. If the net does not exist, the system creates the net, naming it with *`S_netName`*. To associate the shape with a net, this argument is required.
Default: none

`?termName` *`S_termName`*

Symbol or character string specifying the name of the terminal and net with which you want to associate the shape. Enclose character strings in quotation marks (""); precede symbols with a single quotation mark ('). ROD converts symbols to character strings. If the terminal and net does not exist, the system creates them, naming them *`S_termName`*.

If you specify both *`S_termName`* and *`S_netName`*, their names must be the same. If a terminal with the specified name already exists, the net it belongs to must have the same name. If the net name does not match the terminal name, the function reports an error. To associate the shape with a terminal and net, this argument is required.
Default: none

`?termIOType` *`S_termIOType`*

Symbol or character string specifying the direction type (I/O type) for the terminal. If *`S_termName`* is not specified, *`S_termIOType`* is ignored. If a terminal with the name specified by *`S_termName`* already exists, it must have the same direction type as specified by *`S_termIOType`*; if the direction type is not the same, the function reports an error.
Valid Values: `input`, `output`, `inputOutput`, `switch`, `jumper`
Default: `inputOutput`

`?pin` *`g_pin`*

Boolean value indicating whether or not to make the shape into a pin. The value must be `t` or `nil`.

When you specify *`g_pin`*, you must also specify *`S_netName`* and *`S_termName`*; if you do not, all connectivity arguments are ignored but no error is reported.

When the value is `t`, the shape created is a pin. When the value is `nil`, the shape created is not a pin, and all other arguments containing the word `pin` are ignored.
Valid Values: `t`, `nil`
Default: `nil`

`?pinAccessDir` *`tl_pinAccessDir`*

Text string or list specifying the access directions for the pin. Enclose string values in quotation marks. Do not use a symbol. An example of a list is
`list( "top" "bottom" )`
Valid Values: `top`, `bottom`, `left` `right`, `any`, `none`, or a list containing any of these values
Default: `any`

`?pinLabel` *`g_pinLabel`*

Boolean value indicating whether or not to add a text-display label for the pin. The value must be `t` or `nil`. When the value is `t`, the system creates a text-display label attached to the pin. The text is the contents the of the *`S_termName`* argument.
Valid Values: `t`, `nil`
Default: `nil`

`?pinLabelHeight` *`n_pinLabelHeight`*

Integer or floating-point number specifying the vertical height of the pin text-display label in user units.
Default:`1`

`?pinLabelLayer` *`txl_pinLabelLayer`*

Text string, integer, or list specifying the layer or layer-purpose pair for the pin text-display label, such as

```
?pinLabelLayer "metal1"
?pinLabelLayer 45
?pinLabelLayer list("metal1" "drawing")
```

Enclose string values in quotation marks. Do not use a symbol. For lists, use the following format:
`list(`*`layer purpose`*`)`

Examples of layer-purpose pairs are

```
?pinLabelLayer list("metal1" "drawing")
?pinLabelLayer list(45 252)
?pinLabelLayer list(45 "drawing")
?pinLabelLayer list("metal1" 252)
```

When only the layer is specified, the purpose defaults to the purpose defined by *`l_layer`*. When neither layer nor purpose is specified, both layer and purpose default to the values defined by *`l_layer`*.

Default: layer specified by *`l_layer`*; purpose specified by *`l_layer`*

`?pinLabelFont` *`S_pinLabelFont`*

Symbol or character string specifying the name of the font for the pin text-display label.
Valid Values: `euroStyle`, `gothic`, `math`, `roman`, `script`, `stick`, `swedish`
Default: `stick`

`?pinLabelOrient` *`g_pinLabelDrafting`*

Boolean value indicating whether or not to rotate the pin text-display label more than 90 degrees. The value must be `t` or `nil`. When the value is `t`, the label can rotate 90 degrees or less. When the value is `nil`, the label can rotate 90 degrees or more.
Valid Values: `t`, `nil`
Default: `t`

`?pinLabelOrient` *`S_pinLabelOrient`*

Symbol or character string specifying the orientation of the pin text-display label.
Valid Values:

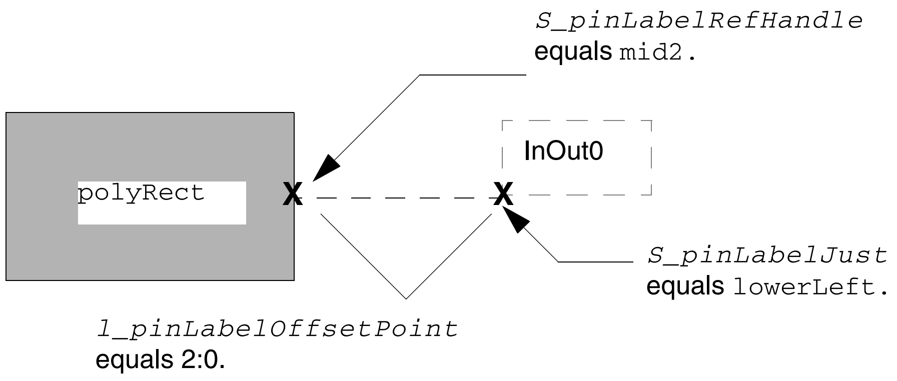
`0`	`R0`	`MY`	`sideways`
`90`	`R90`	`MYR90`	`sideways&90`
`180`	`R180`	`MX`	`upsideDown`
`270`	`R270`	`MXR90`	`sideways&270`

Default: `R0`

`?pinLabelOffsetPoint` *`l_pinLabelOffsetPoint`*

Single set of coordinates specifying the X and Y offset of the origin of the pin text-display label from a point handle on the rectangle. The label origin is specified by *`S_pinLabelJust`*, and the rectangle point handle is specified by *`S_pinLabelRefHandle`*.

For example, to create a label on the rectangle `polyRect`, offset from the segment point handle `mid2` by 2 units in the direction of the X axis and 0 units in the direction of the Y axis, specify 2:0.



Use one of the following formats:
`x``:`*`y`*
or
`list(`*`x y`*`)`
or any expression that evaluates to a list containing two numbers.
Default: `0:0`

`?pinLableJust` *`S_pinLabelJust`*

Symbol or character string specifying the origin point of the label. You must specify both words; do not abbreviate them.

Valid Values:

`upperLeft`	`lowerLeft`
`upperCenter`	`lowerCenter`
`upperRight`	`lowerRight`
`centerLeft`	
`centerCenter`	
`centerRight`	

Default: `centerCenter`

`?pinLabelRefHandle` *`S_pinLabelRefHandle`*

Symbol or character string specifying the point handle on the rectangle with which you want to associate the origin point of the text-display label. You can specify the name of any bounding box or segment point handle.

Bounding box point handle names: To specify a point handle on the bounding box of a rectangle, use either the two-word handle name or its two-character abbreviation, as shown in valid values. For example, you can use `uL` for `upperLeft`.
Valid Values for bounding box point handles:

`upperLeft`	or	`uL`	`lowerLeft`	or	`lL`
`upperCenter`	or	`uC`	`lowerCenter`	or	`lC`
`upperRight`	or	`uR`	`lowerRight`	or	`lR`
`centerLeft`	or	`cL`			
`centerCenter`	or	`cC`			
`centerRight`	or	`cR`			

Segment point handle names: To specify a point handle on a segment of a rectangle, use the system-defined name for the handle. Each segment has three system-defined point handles, named `start`*`n`*, `mid`*`n`*, and `end`*`n`*, where *`n`* is the segment number, starting at zero.

The last segment also has the following point handles: `startLast`, `midLast`, and `endLast`. The system numbers segments clockwise, starting with the lower-left corner as zero, no matter how the rectangle was created.

For a complete description of system-defined point handles, see “System-Defined Handles”.
Valid Values: `start0`, `mid0`, `end0`; `start1`, `mid1`, `end1`; `start2`, `mid2`, `end2`; `startLast`, `midLast`, `endLast`
Default: the bounding box point handle `centerCenter`

Subrectangle Arguments (l_subrectArgs)

`?layer` *`txl_layer`*

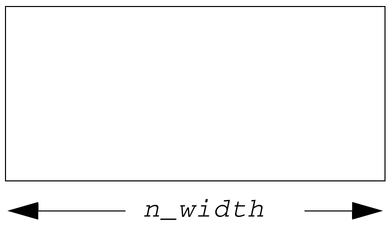
Text string, integer, or list specifying the layer or layer-purpose pair for the subrectangle(s), such as

```
?layer "metal1"
?layer 45
?layer list("metal1" "drawing")
```

Enclose string values in quotation marks. Do not use a symbol. Use one of the formats defined for the argument *`txl_layer`*. You are required to specify a layer.
Default Purpose: `drawing`

`?width` *`n_width`*

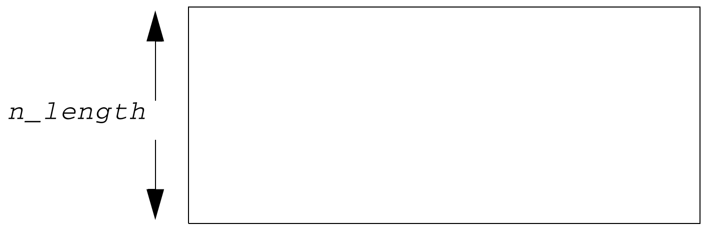
Positive integer or floating-point number specifying the horizontal measurement of the subrectangle(s) in user units.



If not specified, the system uses *`n_length`*; if neither is specified, the system uses the `minWidth` rule for the subrectangle layer from the technology file. If the `minWidth` rule is not defined in the technology file, `rodCreateRect` returns `nil` to indicate an error.
Default: *`n_length`* if specified; otherwise `minWidth` for the subrectangle layer from the technology file

`?length` *`n_length`*

Positive integer or floating-point number specifying the vertical measurement of the subrectangle(s) in user units.



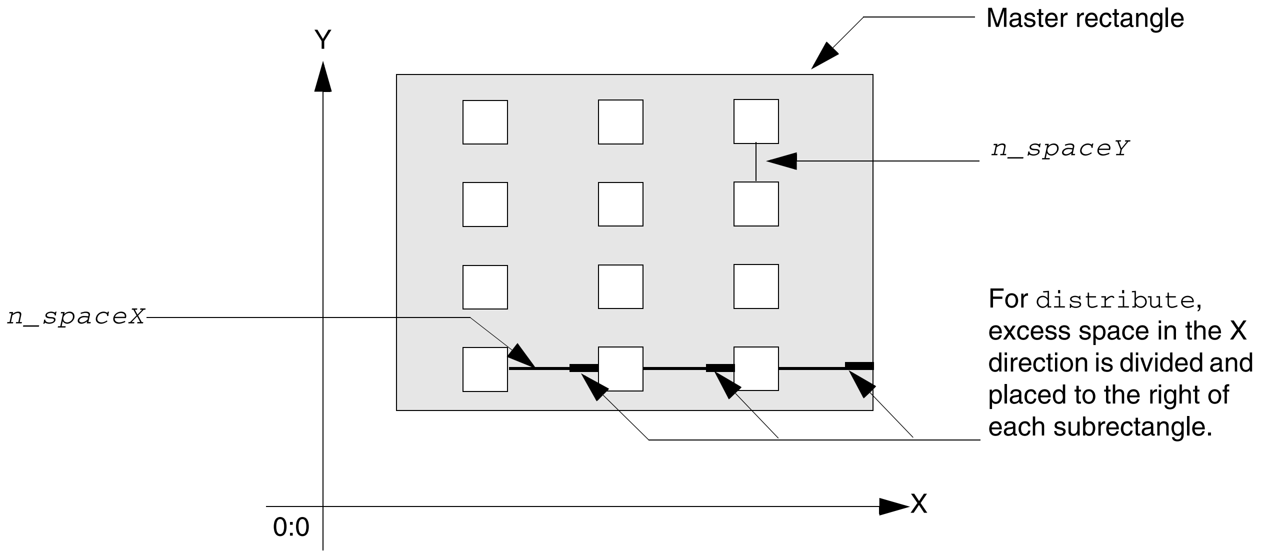
If not specified, the system uses *`n_width`*; if neither is specified, the system uses the `minWidth` rule for the subrectangle layer from the technology file. If the `minWidth` rule is not defined in the technology file, `rodCreateRect` returns `nil` to indicate an error.
Default: *`n_width`* if specified; otherwise, `minWidth` for the subrectangle layer from the technology file

`?gap` *`S_gap`*

Symbol or character string specifying the method the system uses to place subrectangles within each master rectangle. Enclose character strings in quotation marks (""); precede symbols with a single quotation mark ('). ROD converts symbols to character strings.

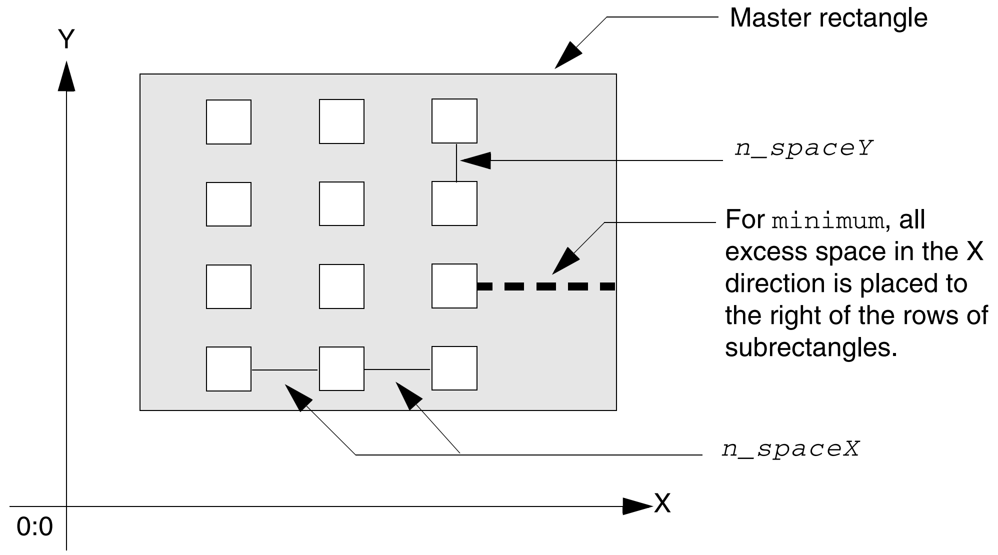
The system automatically calculates the maximum number of rectangles that fit in the master rectangle, allowing for the space needed by the *`offset`* arguments, and using the value of the *`n_spaceX`* and *`n_spaceY`* arguments to determine the minimum space between subrectangles.

When the value of *`S_gap`* is `distribute`, which is the default, the system distributes the space around subrectangles as evenly as possible, in multiples of the grid space specified by `mfgGridResolution`. Any remaining space is divided evenly and placed to the right and/or above each subrectangle. The following example shows excess space along the X axis.

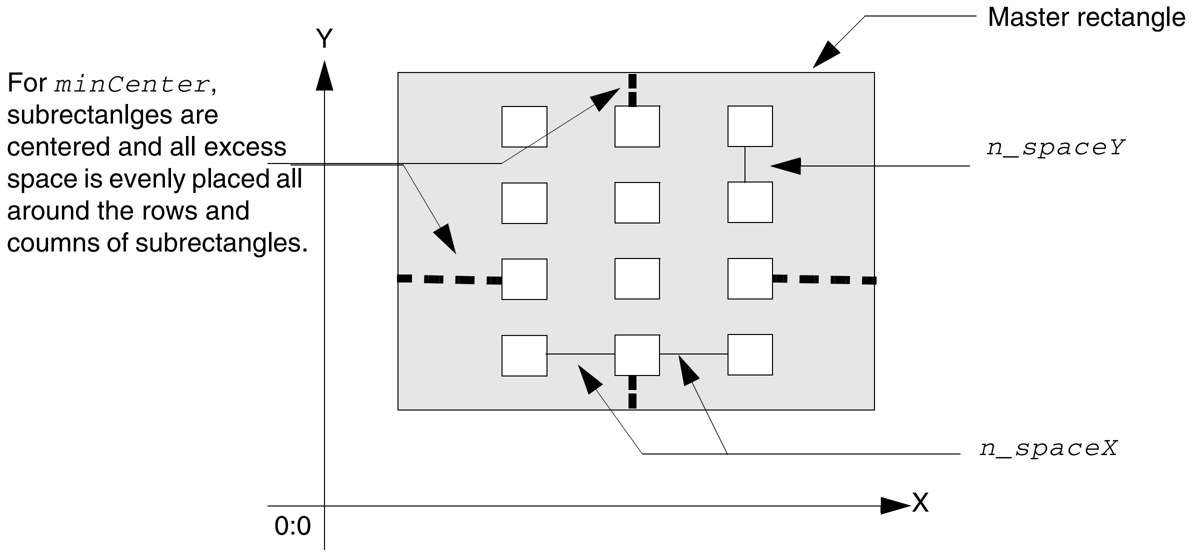

Master rectangle
Y
n_spaceY
n_spaceX
For distribute, excess space in the X direction is divided and placed to the right of each subrectangle.
X
0:0

When the value of *S_gap* is `distribute` and there is only one subrectangle, the system centers the subrectangle in the master rectangle, on grid.

When the value of *S_gap* is `minimum`, the system places subrectangles *n_spaceX* and *n_spaceY* apart until there is no space for another subrectangle, then places all of the excess space to the right and/or above the rows/columns of subrectangles. The following example shows excess space along the X axis.


Master rectangle
Y
n_spaceY
For minimum, all excess space in the X direction is placed to the right of the rows of subrectangles.
n_spaceX
X
0:0

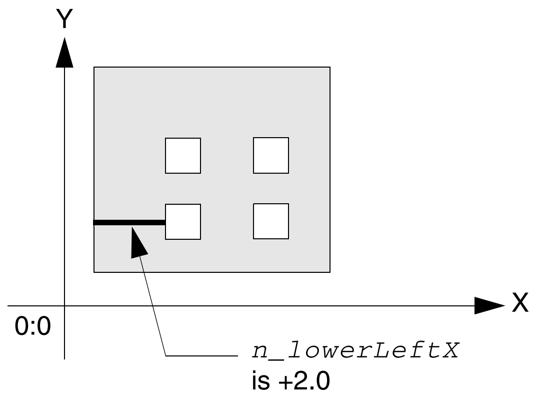
When the value of *`S_gap`* is `minCenter`, the system places subrectangles *`n_spaceX`* and *`n_spaceY`* apart until there is no space for another subrectangle. The subrectangles are then placed in the center by spreading the excess space evenly around these rows and columns of subrectangles. The following example shows excess space along the X axis.

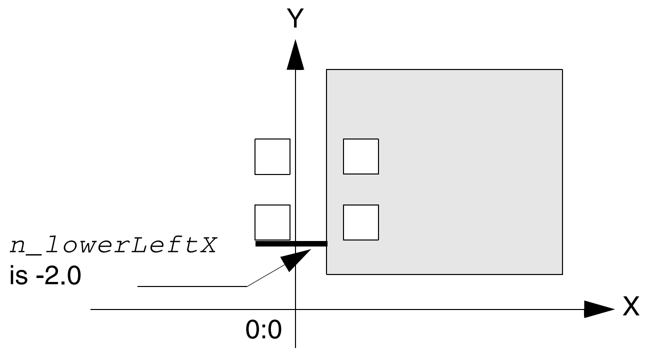

Master rectangle
Y
For *minCenter*, subrectanlges are centered and all excess space is evenly placed all around the rows and coumns of subrectangles.
n_spaceY
n_spaceX
X
0:0

Valid Values: `distribute`, `minimum, minCenter`
Default: `distribute`

`?lowerLeftOffsetX` *`n_lowerLeftOffsetX`*

Signed integer or floating-point number specifying the offset along the X axis of the left edge of the lower-left subrectangle from the left edge of the master rectangle. A positive number starts the subrectangles to the right; a negative number starts the subrectangles outside the master rectangle.

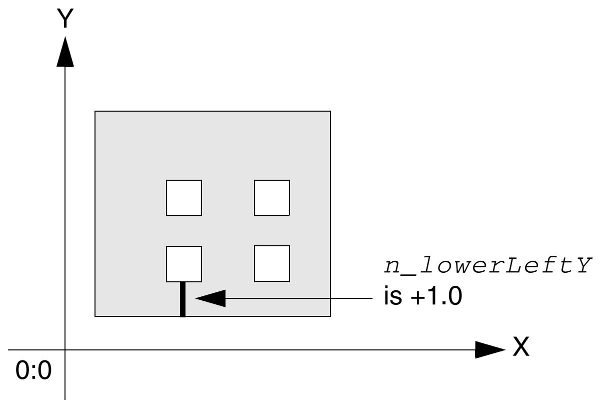


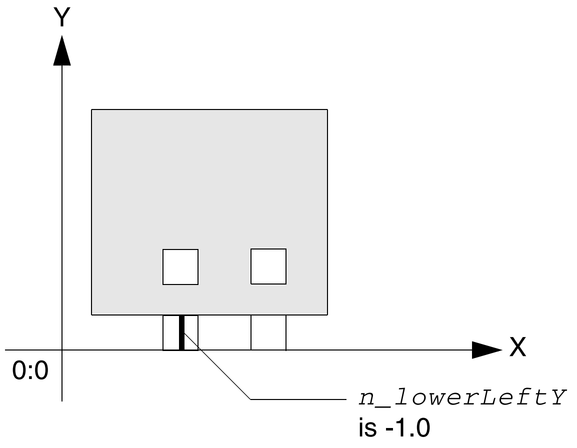


Default: `0`

`?lowerLeftOffseY` *`n_lowerLeftOffsetY`*

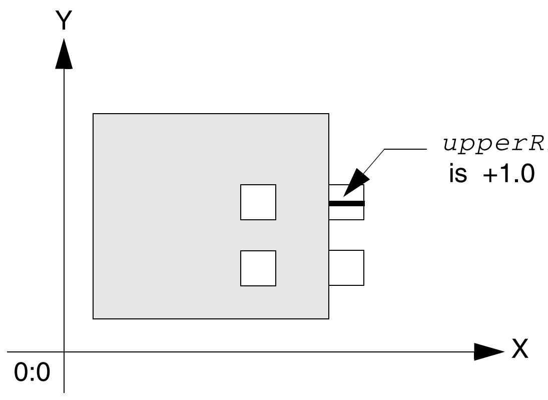
Signed integer or floating-point number specifying the offset along the Y axis of the bottom edge of the lower-left subrectangle from the bottom edge of the master rectangle. A positive number starts the subrectangles above the bottom edge of the master rectangle; a negative number starts the subrectangles outside the master rectangle.

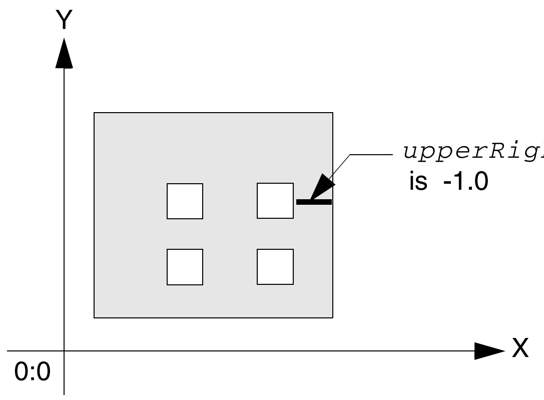




`?upperRightOffsetX` *`n_upperRightOffsetX`*

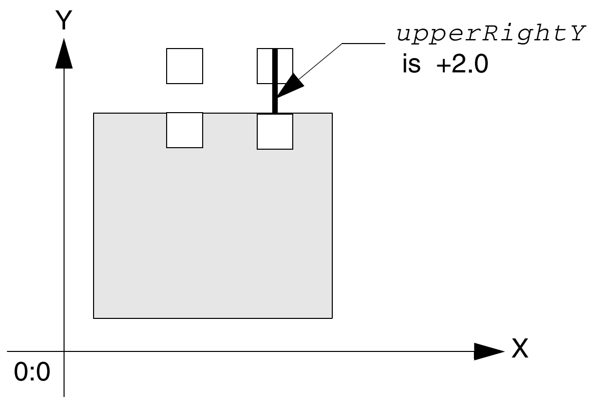
Signed integer or floating-point number specifying the offset along the X axis of the right edge of the upper-right subrectangle from the right edge of the master rectangle. A positive number starts the subrectangles outside of the master rectangle; a negative number starts the subrectangles to the left of the right edge of the master rectangle.

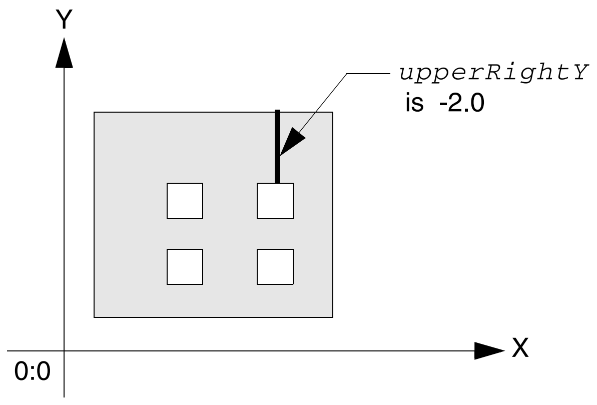




`?upperRightOffsetY` *`n_upperRightOffsetY`*

Signed integer or floating-point number specifying the offset along the Y axis of the top edge of the upper-right subrectangle from the top edge of the master rectangle. A positive number starts the subrectangles outside of the master rectangle; a negative number starts the subrectangles below the top edge of the master rectangle.





`?spaceX` *`n_spaceX`*

Positive integer or floating-point number specifying the distance between the edges of subrectangles in the direction of the X axis. If not specified, the system uses the `minSpacing` rule for the subrectangle layer from the technology file. If the `minSpacing` rule is not defined in the technology file, `rodCreateRect` returns `nil` to indicate an error.
Default: `minSpacing` for the subrectangle layer from the technology file

`?spaceY` *`n_spaceY`*

Positive integer or floating-point number specifying the distance between the edges of subrectangles in the direction of the Y axis. If not specified, the system uses the `minSpacing` rule for the subrectangle layer from the technology file. If the `minSpacing` rule is not defined in the technology file, `rodCreateRect` returns `nil` to indicate an error.
Default: `minSpacing` for the subrectangle layer from the technology file

`?prop` *`l_prop`*

A list or lists of lists, where each list contains the name of a property and its value. When the property name is a string, enclose it in quotes. For a description of the valid data types, see the "About the Add Property Form" in the *Virtuoso Layout Suite L User Guide*. For Boolean properties, the value must be `t` or `nil`. For example, you would specify a single Boolean property as follows:

```
?prop list("myProp" t)
```

and a list of properties as follows:

```
?prop list(
   list("myProp1" propValue1)
   list("myProp2" propValue2)
   ...
   list("myPropN" propValueN)
)
```

For a more detailed description about specifying lists of lists, see Formatting List-of-Lists Arguments for Subparts.
Default: `nil`

Value Returned

The values returned by the `rodCreateRect` function is determined by how you specify the *`g_returnBoolean`* argument.

- When *`g_returnBoolean`* is set to `t`, the function returns `t` or `nil`.
- When *`g_returnBoolean`* is set to `nil`, the function returns a ROD object ID or list of ROD object IDs.
- When *`g_returnBoolean`* is set to `nil` and more than one rectangle is created, `rodCreateRect` returns a list of ROD object IDs with the upper right rectangle listed first and the lower left rectangle listed last. For example,

```
("xxx.6.6" "xxx.6.5" "xxx.6.4" "xxx.6.3" "xxx.6.2" "xxx.6.1"
 "xxx.5.6" "xxx.5.5" "xxx.5.4" "xxx.5.3" "xxx.5.2" "xxx.5.1"
 "xxx.4.6" "xxx.4.5" "xxx.4.4" "xxx.4.3" "xxx.4.2" "xxx.4.1"
 "xxx.3.6" "xxx.3.5" "xxx.3.4" "xxx.3.3" "xxx.3.2" "xxx.3.1"
 "xxx.2.6" "xxx.2.5" "xxx.2.4" "xxx.2.3" "xxx.2.2" "xxx.2.1"
 "xxx.1.6" "xxx.1.5"  "xxx.1.4" "xxx.1.3" "xxx.1.2"  "xxx.1.1")
```

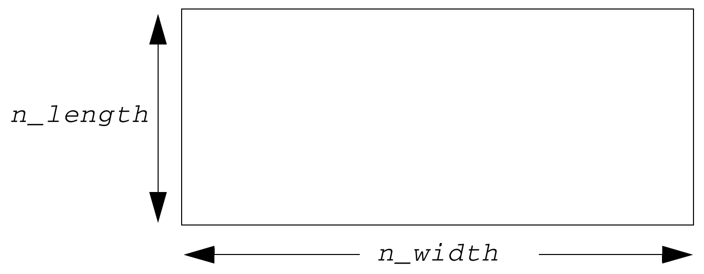
`R_rodObj`	ROD object ID or a list of ROD object IDs in descending sequence for ROD object information associated with the new rectangle or multiple new rectangles.
`t`	Rectangles were created successfully.
`nil`	An error occurred, and no rectangles were created.

Examples

Example 1: Creating a Single Rectangle

Note: When you want to create a rectangle that is a regular, unnamed database shape, use the `dbCreateRect` function instead of the `rodCreateRect` function; using `dbCreateRect` produces less overhead, resulting in faster performance.

You can create a single named rectangle, with ROD attributes, as shown below:



with code similar to the following:

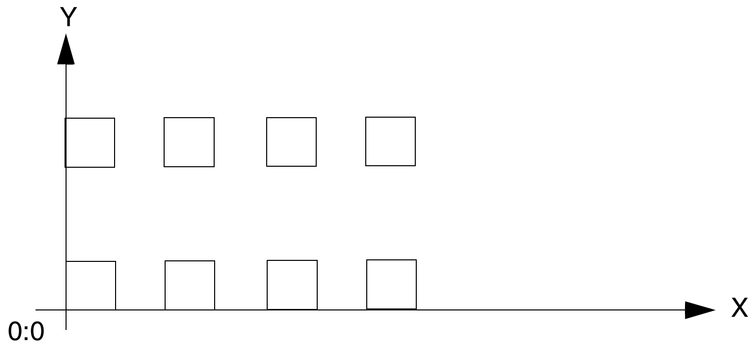
```
rodCreateRect(
    ?layer    "metal2"
    ?width    4
    ?length   2
)
```

The return value is a single ROD object ID.

Example 2: Creating Rows and Columns of Named Rectangles

To create rows and columns of rectangles that are regular, unnamed database shapes, see Filling a Bounding Box with Rectangles.

You can create rows and columns of named rectangles that have ROD attributes, as shown below:



with code similar to the following.

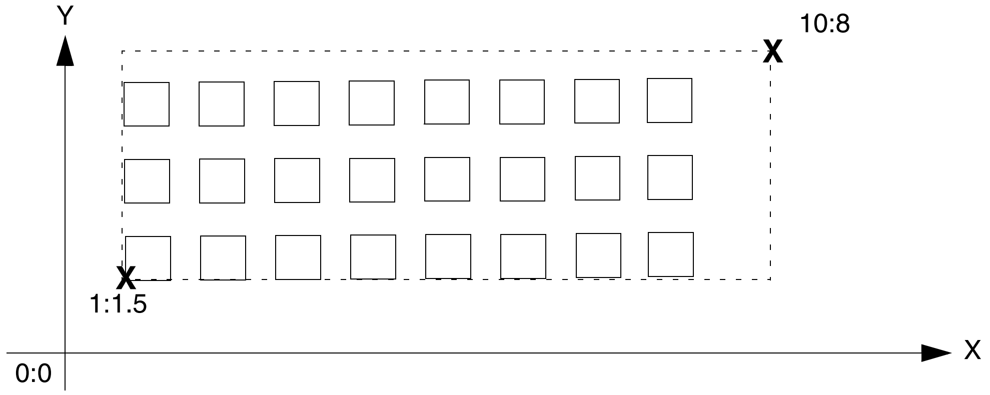
```
rodCreateRect(
    ?layer        "metal1"
    ?width         1
    ?length        1
    ?origin        list( 0 0 )
    ?elementsX     4
    ?elementsY     2
     ?spaceX       1
    ?spaceY        2
)
```

The return value is a list of eight ROD object IDs.

Example 3: Filling a Bounding Box with Named Rectangles

To fill a bounding box with rectangles that are regular, unnamed database shapes, see Filling a Bounding Box with Rectangles.

You can fill a bounding box with named rectangles that have ROD attributes, as shown below:



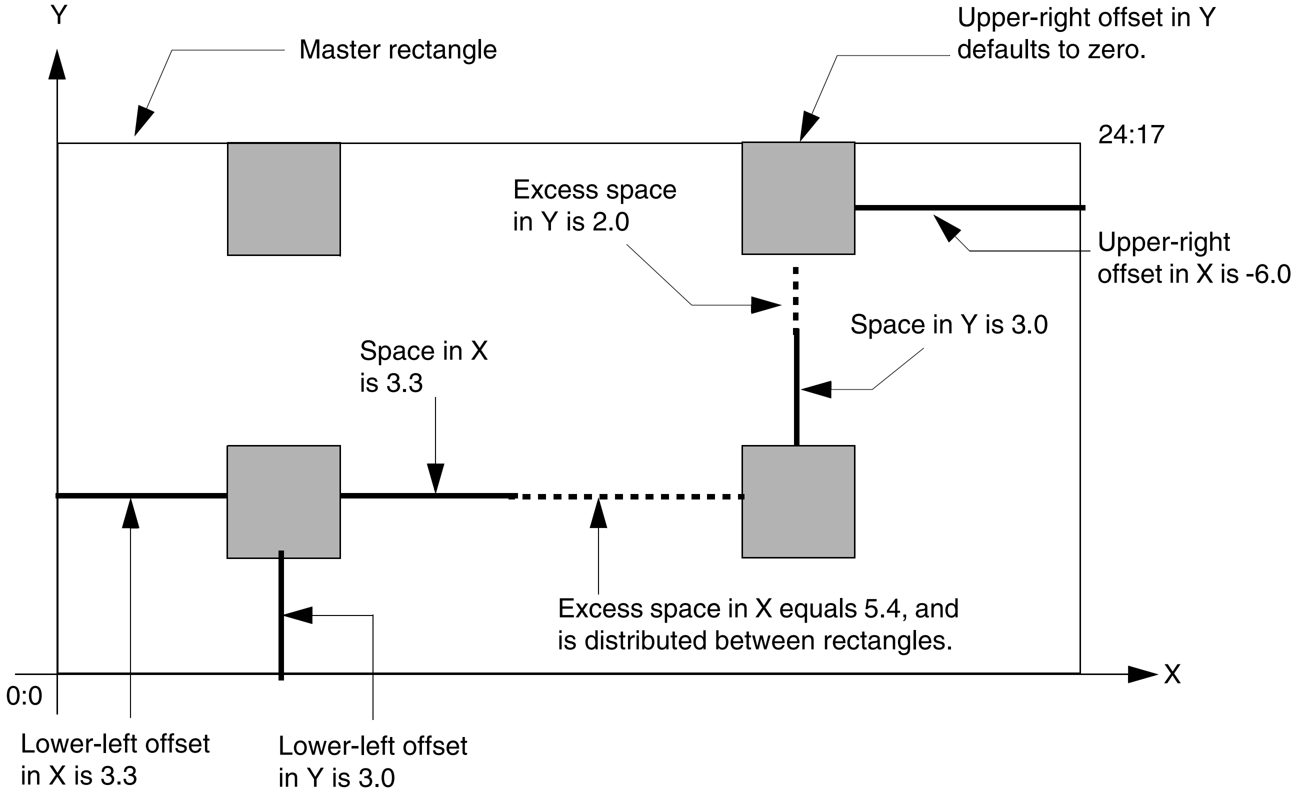
with code similar to the following (the first rectangle is always in the lower-left corner of the bounding box):

```
rodCreateRect(
    ?layer          "metal1"
    ?fillBBox       list( 1:1.5 10:5 )
    ?spaceX         0.5
    ?returnBoolean  t
)
```

The return value is `t`.

Example 4: Creating a Multipart Rectangle

You can create a multipart rectangle consisting of a single master rectangle filled with unnamed subrectangles as shown below:

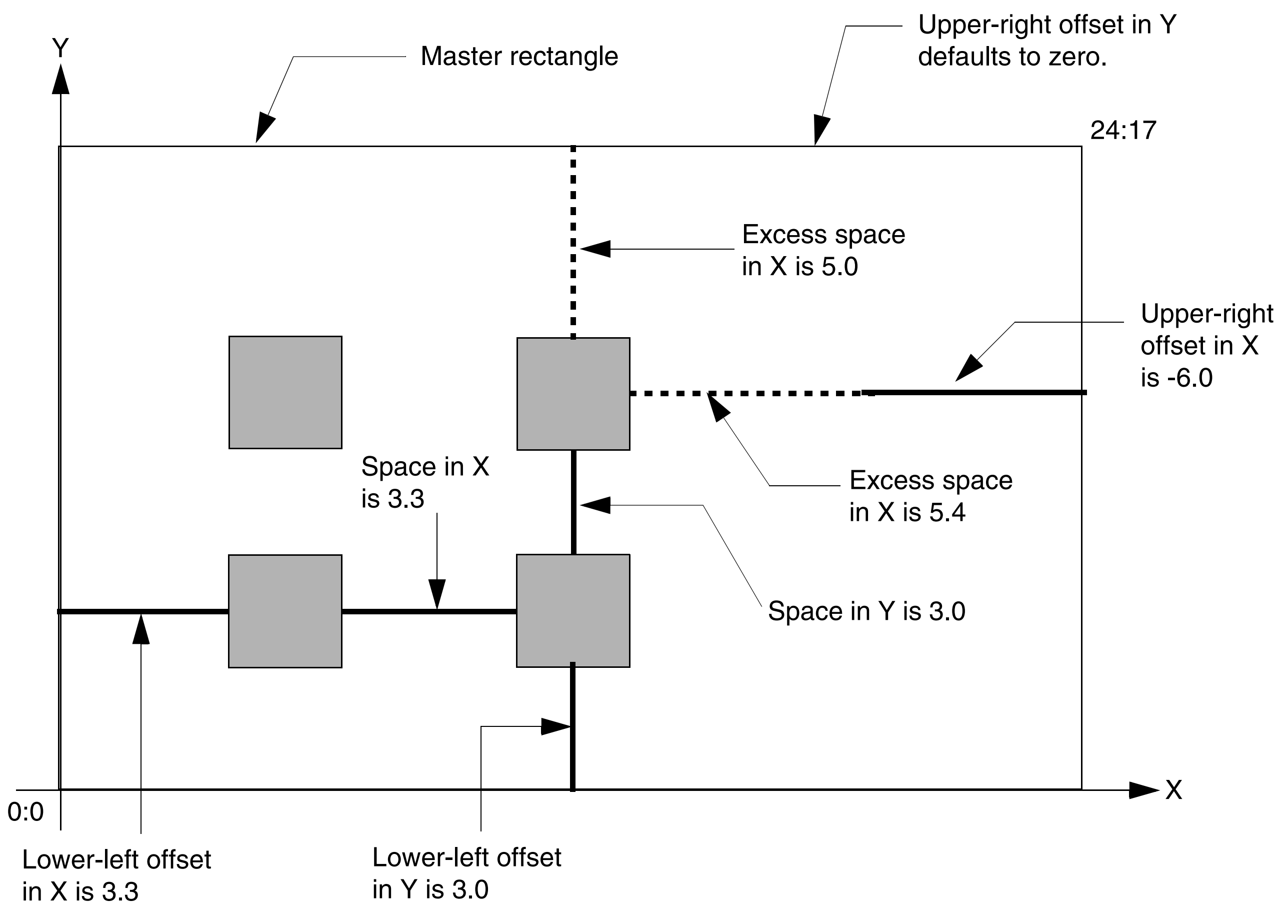

Y
Master rectangle
Upper-right offset in Y defaults to zero.
24:17
Excess space in Y is 2.0
Upper-right offset in X is -6.0
Space in Y is 3.0
Space in X is 3.3
Excess space in X equals 5.4, and is distributed between rectangles.
0:0
X
Lower-left offset in X is 3.3
Lower-left offset in Y is 3.0

with code similar to the following. In this example, *S_gap* is not specified; it defaults to `distribute`, causing the excess space to be placed between the subrectangles.

```
cvId = deGetCellView()

rodCreateRect(
    ?layer          "metal1"
    ?width          24
    ?length         17
    ?origin         list( 0 0 )
    ?elementsX      1
    ?elementsY      1
    ?cvId           cvId
    ?subRectArray
    list(
           list(
                  ?layer                "metal2"
                  ?width                3
                  ?length               3
                  ?lowerLeftOffsetX     3.3
                  ?lowerLeftOffsetY     3
                  ?upperRightOffsetX    -6
                  ?spaceX               3.3
                  ?spaceY               3
        );end first subrectangle list
        );end all subrectangle lists
    ) ;end rodCreateRect
```

Specifying the Gaps as Minimum

To place excess space to the right or above the subrectangles, you can specify the *`S_gap`* argument as `minimum`, as shown below.


Y
Master rectangle
Upper-right offset in Y defaults to zero.
24:17
Excess space in X is 5.0
Upper-right offset in X is -6.0
Space in X is 3.3
Excess space in X is 5.4
Space in Y is 3.0
X
0:0
Lower-left offset in X is 3.3
Lower-left offset in Y is 3.0

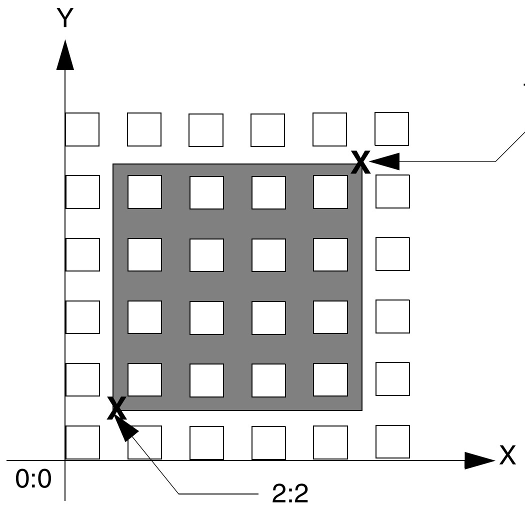
300
84
500
298
1250
1250
300
300

```
cvId = deGetCellView()
```

```
rodCreateRect(
        ?layer         "metal1"
        ?width         24
        ?length        17
        ?origin        list( 0 0 )
        ?elementsX     1
        ?elementsY     1
        ?cvId          cvId
         ?subRectArray
        list(
               list(
                      ?layer            "metal2"
                      ?width            3
                      ?length           3
                      ?gap              "minimum"
                      ?lowerLeftOffsetX  3.3
                      ?lowerLeftOffsetY  3
                      ?upperRightOffsetX -6
                      ?spaceX            3.3
                      ?spaceY            3
                   ) ;End first subrectangle list
             ) ;end all subrectangle lists
        ) ; end rodCreateRect
```

Example 5: Creating a Multipart Rectangle with Overlapping Subrectangles

You can create a multipart rectangle with overlapping unnamed subrectangles, as shown below:



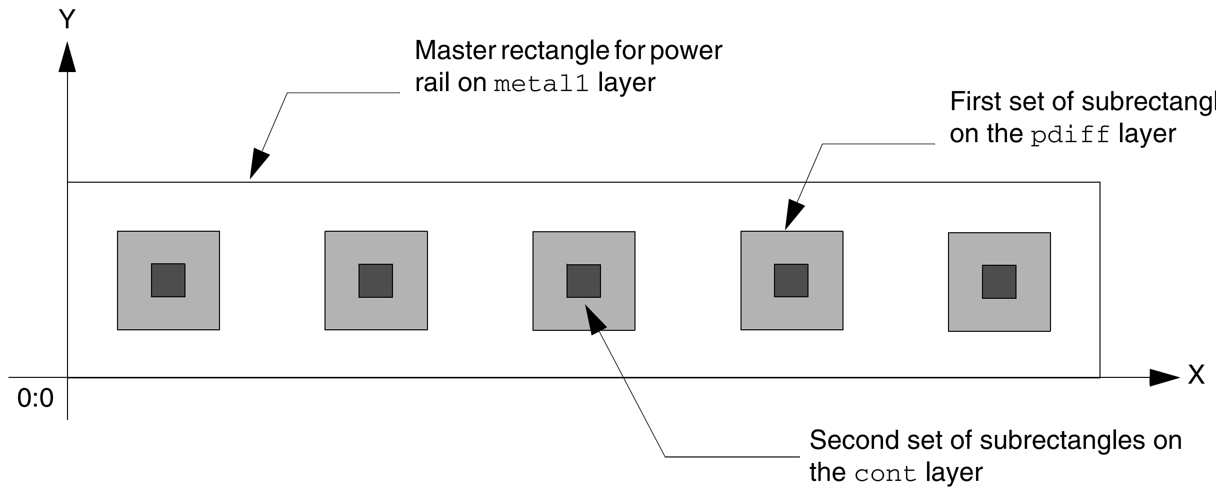
with code similar to the following;, where the lower-left offset arguments are less than zero and the upper-right offset arguments are greater than zero:

```
cvId = deGetCellView()
```

```
rodCreateRect(
    ?layer         "metal1"
    ?width         12
    ?length         12
    ?origin         list( 2 2 )
    ?elementsX      1
    ?elementsY      1
    ?cvId           cvId
     ?subRectArray
          list(
             list(
                  ?layer                "metal2"
                  ?width                1
                  ?length               1
                  ?lowerLeftOffsetX     -2
                  ?lowerLeftOffsetY     -2
                  ?upperRightOffsetX    2
                  ?upperRightOffsetY    2
                  ?spaceX               1
                  ?spaceY               1
                ) ;end first subrectangle list
         );end all subrectangle lists
    );end rodCreateRect
```

Example 6: Creating a Ground Rail and Contacts

You can create a ground rail with contacts, as shown below:


Y
Master rectangle for power rail on metal1 layer
First set of subrectangles on the pdiff layer
X
0:0
Second set of subrectangles on the cont layer

where with code similar to the following, you specify a named rectangle for the metal layer, a set of unnamed subrectangles on the p-diffusion layer, and a second set of unnamed subrectangles on the contact layer:

```
rodCreateRect(
    ?layer          "metal1"
    ?width          10
    ?length         2
    ?origin         list( 0 0 )
    ?elementsX      1
    ?elementsY      1
    ?cvId           cvId
    ?subRectArray
```

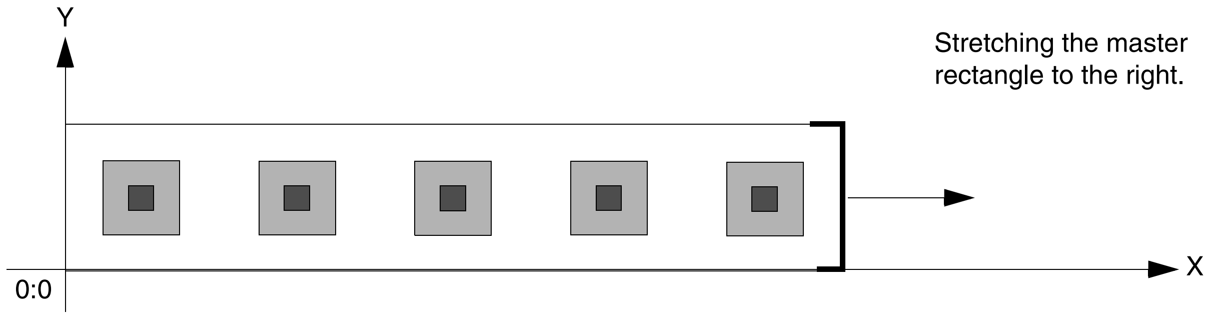
```
        list(
            list(
                ?layer                "pdiff"
                ?width                1
                ?length               1
                ?gap                  "minimum"
                ?lowerLeftOffsetX     0.5
                ?lowerLeftOffsetY      0.5
            ) ;End of first subrectangle list
            list(
                ?layer                "cont"
                ?width                0.5
                ?length               0.5
                ?gap                  "minimum"
                ?lowerLeftOffsetX     0.75
                ?lowerLeftOffsetY     0.75
                ?spaceX               1.5
                ?spaceY               1.5
            ) ;end second subrectangle list
        ) ;end all subrectangle lists
    ) ; end rodCreateRect
```

In this example, specifying the space arguments for the subrectangles is not necessary; the software uses the default values for the *`n_spaceX`* and *`n_spaceY`* arguments to create the number of contacts that fit. However, if you want to an amount of space between contacts that is different than the default when the power rail is stretched, you do need to specify the space arguments.

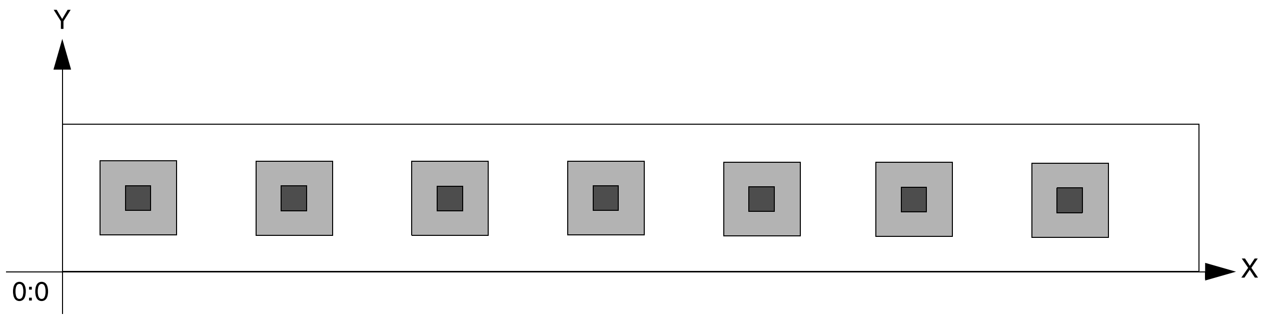
Example 7: Stretching a Multipart Rectangle

The following examples include stretching the master rectangle of a multipart rectangle in the direction of the Y axis. To make the subrectangles regenerate correctly, specifying *`n_spaceY`* is necessary. The space arguments default to the minimum spacing (`minSpacing`) defined for the subrectangle layers in the technology file, which, in this case, does not produce the desired result.

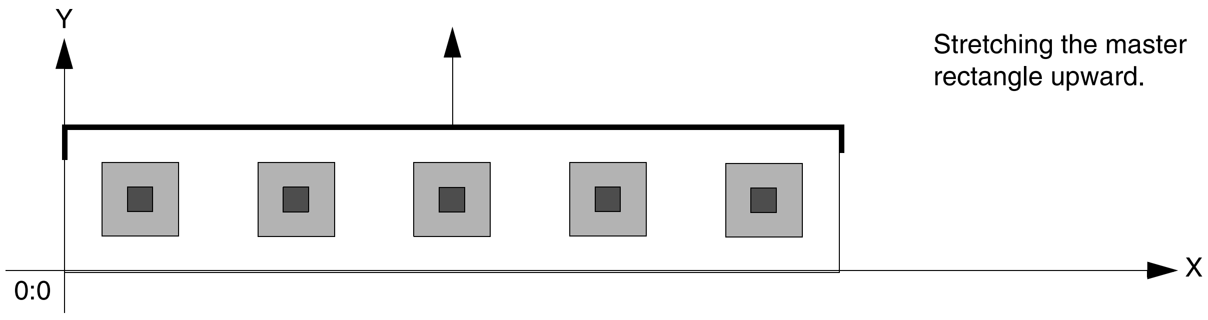
When you stretch a master rectangle using the Virtuoso® Layout Editor *Edit – Stretch* command, the system regenerates all sets of unnamed subrectangles. For example, stretching the master rectangle specified in the last example to the right, as shown below:



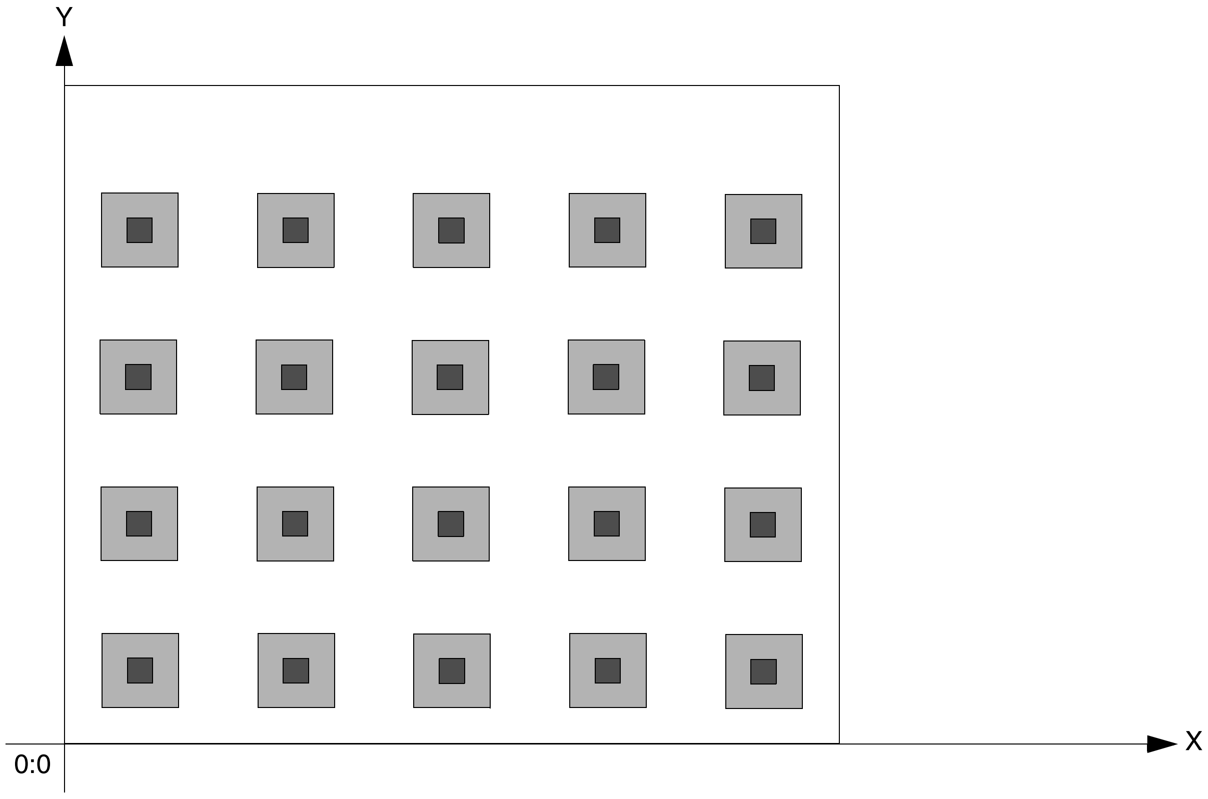
makes the power rail longer, adding more contacts horizontally, like this:



Stretching the master rectangle upward, as shown below:

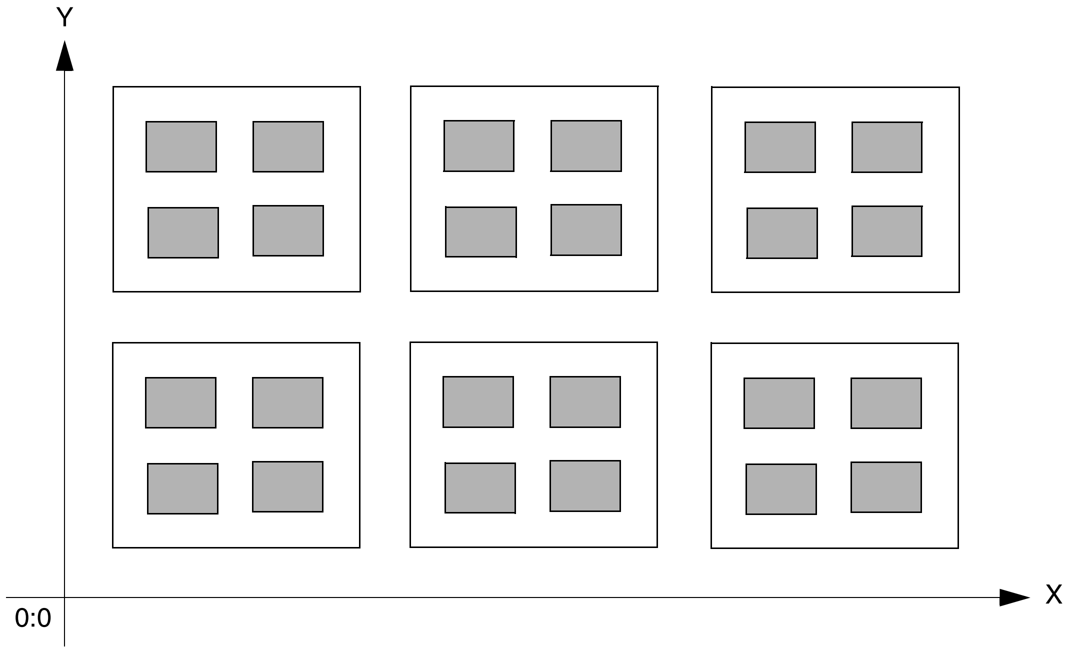


makes the power rail wider, adding more rows of contacts vertically, like this:



Example 8: Creating Multiple Master Rectangles Filled with Subrectangles

For a multipart rectangle, you can create rows and columns of master rectangles and fill them with unnamed subrectangles, as shown below:

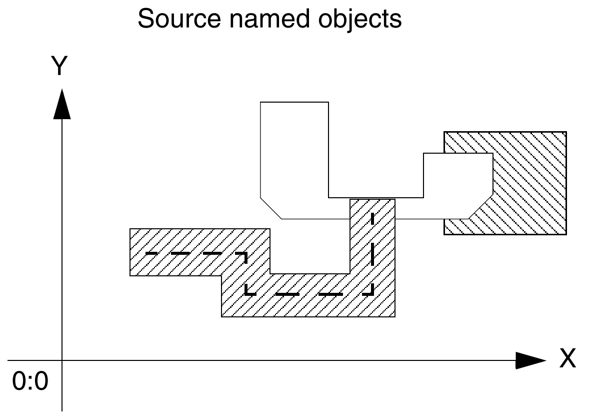


with code similar to the following:

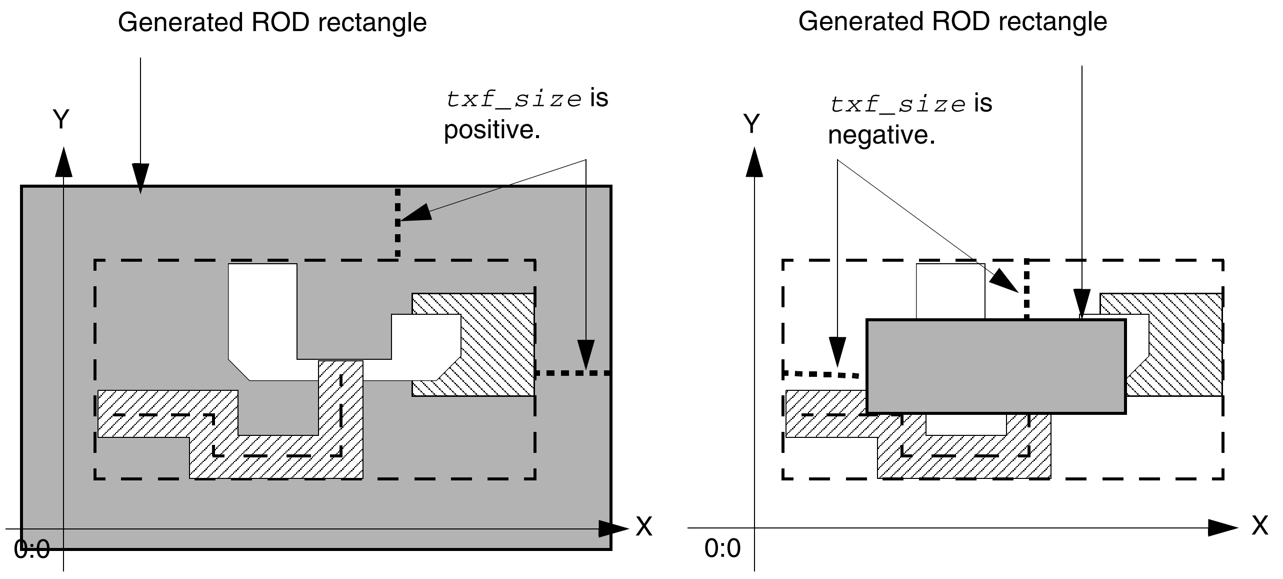
```
cvId = deGetCellView()

rodCreateRect(
    ?layer          "metal1"
    ?width          3.5
    ?length         3
    ?origin         list( 0.75 0.75 )
    ?elementsX      3
    ?elementsY      2
    ?spaceX         0.75
    ?spaceY         0.75
    ?cvId           cvId
    ?subRectArray
    list(
           list(
                  ?layer                "metal2"
                  ?width                1
                  ?length               0.75
                  ?lowerLeftOffsetX     0.5
                  ?lowerLeftOffsetY     0.5
                  ?upperRightOffsetX    -0.5
                  ?upperRightOffsetY    -0.5
                  ?spaceX               0.5
                  ?spaceY               0.5
           );end first subrectangle list
        ) ;end all subrectangle lists
   ) ;end rodCreateRect
```

Example 9: Creating a Rectangle from Other Objects

You can specify the coordinates of a named rectangle with the `n_width`, `n_length`, and `l_origin` arguments or with the `l_bBox` argument. Optionally, you can use the `Rl_fromObj` argument to specify one or more named objects from which to generate a named rectangle. When you specify two or more named objects as the source for creating a rectangle, as shown in the example below:



the system creates the rectangle based on a bounding box around the objects. The new rectangle is larger or smaller than the bounding box, depending on whether you specify a positive or negative size.



Note: After a new rectangle is generated, no relationship exists between the generated rectangle and its source objects.

For a detailed overview of creating rectangles from other objects, see Creating a Rectangle from Another Object.

rodDeleteHandle

```
rodDeleteHandle(
     R_rodObj
     S_name
     );end rodDeleteHandle
     => t / nil
```

Description

Deletes a user-defined handle. You provide the ROD object ID and handle name.

Arguments

R_rodObj	ROD object ID of the named object for which you want to delete a handle. This argument is required.
S_name	Symbol or character string specifying the name of the handle to delete. Enclose character strings in quotation marks (""); precede symbols with a single quotation mark ('). ROD converts symbols to character strings. You cannot delete a system-defined handle. This argument is required. Valid Values: the name of any existing user-defined handle Default: none

Value Returned

`t`	The handle was deleted successfully.
`nil`	An error occurred and no handle is deleted.

rodDeleteSubPart

```
rodDeleteSubPart(
     R_rodObjId
     t_subpartName
     );end rodDeleteSubPart
     => t / nil
```

Description

Deletes the named subpart from the given multipart path (MPP).

Arguments

`R_rodObjId`	ROD object ID associated with the multipart path (MPP).
`t_subpartName`	String specifying the name of the subpart to be deleted.

Value Returned

`t`	The subpart was deleted successfully.
`nil`	An error occurred and the subpart was not deleted.

Example

```
rodDeleteSubPart(mppId, "subpart0")
```

Deletes the subpart named `subpart0` from the MPP corresponding to `mppId`.

rodFillBBoxWithRects

```
rodFillBBoxWithRects(
     [ ?cvId d_cvId ]
     [ ?layer txl_layer ]
     [ ?fillBBox l_fillBBox ]
     [ ?width n_width ]
     [ ?length n_length ]
     [ ?gap S_gap ]
     [ ?spaceX n_spaceX ]
     [ ?spaceY n_spaceY ]
     [ ?prop l_prop ]
     [ ?returnBoolean g_returnBoolean ]
     );end rodFillBBoxWithRects
     => d_dbId / t / nil
```

Description

Fills a bounding box with rectangles, as many as fit within the bounding box you specify. The rectangles are ordinary unnamed shapes, identified by database IDs; the rectangles have no ROD attributes. You can assign a property name and value, or a list of property names and values, to the set of rectangles. The property or list of properties apply to every rectangle in the bounding box. You specify the return value of the function to be Boolean or a list of the database IDs for the rectangles. When you want to create a one- or two-dimensional array of rectangles, and the rectangles do not need to be ROD objects, use this function instead of rodCreateRect because regular database shapes require less overhead, contributing to faster performance.

Arguments

`?cvId` *`d_cvId`*

Database ID for the cellview in which you are creating rectangles.
Valid Values: valid cellview ID
Default: nil

`?layer` *`txl_layer`*

Text string, integer, or list specifying the layer or layer-purpose pair for the rectangle(s), such as:

```
?layer "metal1"
?layer 45
?layer list("metal1" "drawing")
```

Enclose string values in quotation marks. Use one of the formats defined for the argument *`txl_layer`*. You are required to specify a layer.
Default Purpose: `drawing`

`?fillBBox` *`l_fillBBox`*

List of two points defining opposite corners of a bounding box to fill with rectangles.
Valid Values: list of two points
Default: `nil`

`?width` *`n_width`*

Positive integer or floating-point number specifying the horizontal measurement of the rectangle(s) in user units.

If not specified, the system uses *`n_length`*; if neither is specified, the system uses the `minWidth` rule for the rectangle layer from the technology file. If the `minWidth` rule is not defined in the technology file, `rodFillBBoxWithRects` returns `nil` to indicate an error.
Default: *`n_length`* if specified; otherwise `minWidth` for the rectangle layer from the technology file

`?length` *`n_length`*

Positive integer or floating-point number specifying the vertical measurement of the rectangle(s) in user units.

If not specified, the system uses *`n_width`*; if neither is specified, the system uses the `minWidth` rule for the rectangle layer from the technology file. If the `minWidth` rule is not defined in the technology file, `rodFillBBoxWithRects` returns `nil` to indicate an error.
Default: *`n_width`* if specified; otherwise, `minWidth` for the rectangle layer from the technology file

`?gap` *`S_gap`*

Symbol or character string specifying the method the system uses to place rectangles within the bounding box. Enclose character strings in quotation marks (""); precede symbols with a single quotation mark ('). ROD converts symbols to character strings.

The system always uses the value of `n_spaceX` and `n_spaceY` for minimum space between rectangles and calculates the maximum number of rectangles that fit in the bounding box.

When the value of `S_gap` is `distribute`, the system distributes the space around rectangles as evenly as possible, in multiples of the grid space specified by `mfgGridResolution`. Any remaining space is placed to the right of the rows of rectangles and/or above the columns of rectangles.

Note: When the value of `S_gap` is `distribute` and there is only room for one rectangle, the system centers the rectangle in the bounding box, on grid.

When the value of `S_gap` is `minimum`, the system places rectangles `n_spaceX` and `n_spaceY` apart until there is no space for another rectangle, and then places remaining space to the right of the rows of rectangles and/or above the columns of rectangles.

When the value of `S_gap` is `minCenter`, the system places rectangles `n_spaceX` and `n_spaceY` apart until there is no space for another rectangle. The rectangles are then placed in the center by spreading the excess space evenly around these rows and columns of rectangles.

Valid Values: `distribute`, `minimum, minCenter`
Default: `distribute`

`?spaceX` *`n_spaceX`*

Positive integer or floating-point number specifying the distance between the edges of rectangles in the direction of the X axis. If not specified, the system uses the `minSpacing` rule for the rectangle layer from the technology file.

If the `minSpacing` rule is not defined in the technology file, `rodFillBBoxWithRects` returns `nil` to indicate an error.
Default: `minSpacing` for the rectangle layer from the technology file

`?spaceY` *`n_spaceY`*

Positive integer or floating-point number specifying the distance between the edges of rectangles in the direction of the Y axis. If not specified, the system uses the `minSpacing` rule for the rectangle layer from the technology file. If the `minSpacing` rule is not defined in the technology file, `rodFillBBoxWithRects` returns `nil` to indicate an error.
Default: `minSpacing` for the rectangle layer from the technology file

`?prop` *`l_prop`*

A list or lists of lists, where each list contains the name of a property and its value. When the property name is a string, enclose it in quotes. For a description of the valid data types, see the “About the Add Property Form” in the *Virtuoso Layout Suite L User Guide*. For Boolean properties, the value must be `t` or `nil`. For example, you would specify a single Boolean property as follows:

```
?prop list("myProp" t)
```

```
and a list of properties as follows:
```

```
?prop list(
   list("myProp1" propValue1)
   list("myProp2" propValue2)
   ...
   list("myPropN" propValueN)
)
```

For a more detailed description about specifying lists of lists, see Formatting List-of-Lists Arguments for Subparts.
Default: `nil`

`?returnBoolean` *`g_returnBoolean`*

Boolean value indicating whether or not the `rodFillBBoxWithRects` function returns a Boolean value or list of database object IDs. (This functions does not return a ROD object ID).

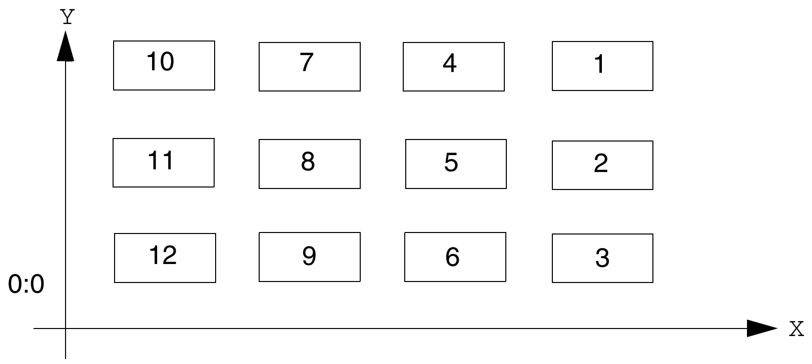
The value of the *g_returnBoolean* argument must be `t` or `nil`. When *g_returnBoolean* is `t`, the function returns `t` or `nil`. When *g_returnBoolean* is `nil`, the function returns a list of database object IDs. When you are creating multiple rectangles, this function executes much more quickly if *g_returnBoolean* is set to `t`, so when you do not need the database object IDs, set this argument to `t`.
Default: `nil`

Value Returned

The value returned by the `rodFillBBoxWithRects` function is determined by how you specify the *g_returnBoolean* argument.]

- ❑ When *g_returnBoolean* is set to `t`, the function returns `t` or `nil`.
- ❑ When *g_returnBoolean* is set to `nil`, the function returns a database ID or list of database IDs. (This function does not return ROD object IDs.)
- ❑ When *g_returnBoolean* is set to `nil` and more than one rectangle is created, `rodFillBBoxWithRects` returns a list of database IDs with the upper right rectangle listed first and the lower left rectangle listed last.

The numbers in the illustration below show the sequence in which the database IDs of the rectangles would be returned.

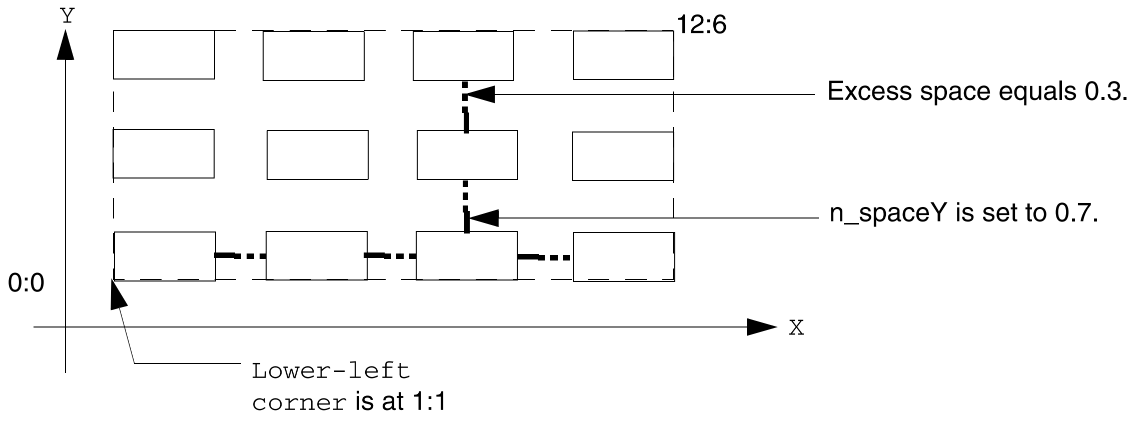


d_dbId	Database object ID or list of database object IDs (**not** the ROD object ID) for the rectangles created to fill the bounding box, in descending sequence.
`t`	Rectangles were created successfully.
`nil`	An error occurred, and no rectangles were created.

Additional Information

You can fill a bounding box with rectangles as shown below by defining the lower-left and upper-right points of the bounding box. Notice where the system places any excess space.

Figure 1-5 Filling a Bounding Box with Rectangles

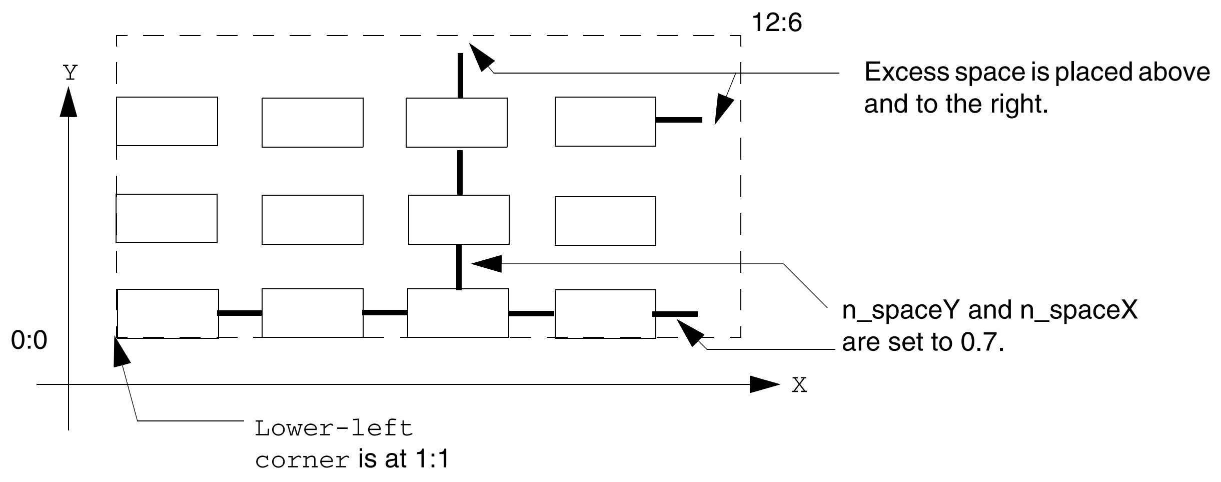


```
cvId = deGetCellView()

dbIdList = rodFillBBoxWithRects(
    ?cvId               cvId
    ?layer              "metal1"
    ?fillBBox           list( 1:1 12:6 )
    ?width              2.0
    ?length             1.0
    ?spaceX             0.7
    ?spaceY             0.7
    ?returnBoolean      nil
    ) ; end rodFillBBoxWithRects
```

In the example above, the *`S_gap`* argument is not specified, so it defaults to `distribute`, causing excess space to be placed between the subrectangles. Therefore, the space between rectangles along the Y axis is 0.3 more than the minimum of 0.7 specified by the *`n_spaceY`* argument. The example code returns the variable `dbIdList`, which contains a list of 12 database IDs.

You can fill the same bounding box specified above, but with the *`S_gap`* argument set to `minimum`, to create the rectangles shown below.

Figure 1-6 Filling a Bounding Box with Rectangles with Minimum Gaps


12:6
Y
Excess space is placed above and to the right.
n_spaceY and n_spaceX are set to 0.7.
0:0
X
Lower-left corner is at 1:1

```
cvId = deGetCellView()

rc = rodFillBBoxWithRects(
        ?cvId               cvId
        ?layer              "metal1"
        ?fillBBox           list( 1:1 12:6 )
        ?width              2.0
        ?length             1.0
        ?gap                "minimum"
        ?spaceX             0.7
        ?spaceY             0.7
        ?returnBoolean      t
    ) ; end rodFillBBoxWithRects
```

The *`S_gap`* argument was set to `minimum`, so the system spaced the rectangles apart using the values of *`n_spaceX`* and *`n_spaceY`*, and placed the remaining space above and to the right of the rectangles.The example code returns the variable `rc`, which is set to `t`.

rodFillWithRects

```
rodFillWithRects(
     [ ?shapeId d_shapeID ]
     [ ?fillShapeLPP lpp ]
     [ ?fillShapeWidth n_width ]
     [ ?fillShapeLength n_length ]
     [ ?gap S_gap ]
     [ ?justification S_justification ]
     [ ?keepOut l_ptList ]
     [ ?spaceX n_spaceX ]
     [ ?spaceY n_spaceY ]
     [ ?enclosureX n_encX ]
     [ ?enclosureY n_encY ]
     [ ?grid n_grid ]
     [ ?outputFillShape l_dbId ]
     );end rodFillWithRects
     => t / nil
```

Description

Fills a base shape with rectangles. The base shape can be a rectangle, polygon or a path type.

Figure 1-7 Symmetric Shape Filling

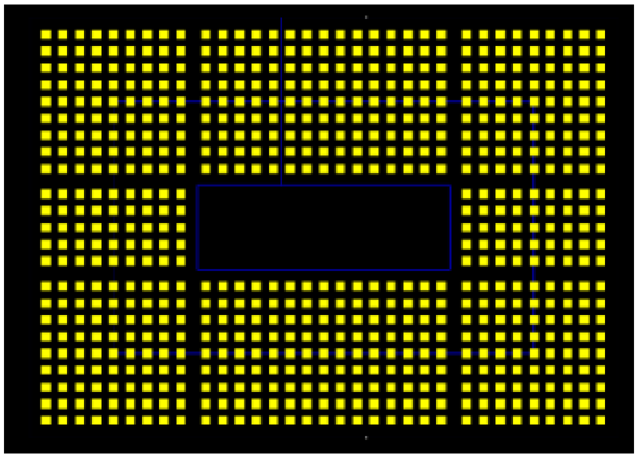


Figure 1-8 Asymmetric Shape Filling

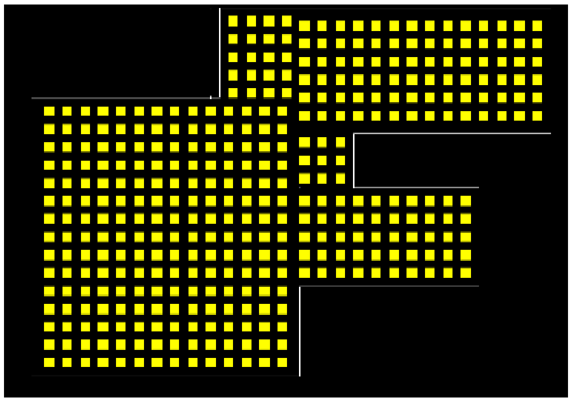
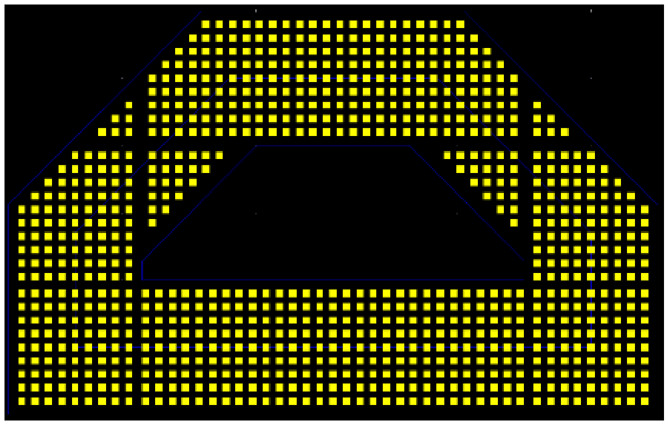


Figure 1-9 Diagonal Filling



Arguments

`?shapeId` *`d_shapeID`*

Database ID of an unnamed base shape or database ID of a ROD object. This shape can be a rectangle, polygon or path type. Other shapes like ellipse or donut are not supported.

`?fillShapeLPP` *`lpp`*

Specifies the layer-purpose pair of the shape that is used for filling the base shape, such as, "`Cont`" or "`drawing`". It is mandatory to specify the value of layer and purpose as a list. For example:

```
?fillShapeLPP list("Cont" "drawing")
```

`?fillShapeWidth` *`n_width`*

Positive integer or floating-point number specifying the horizontal measurement or width of the fill shape in user units.

Note: The only valid shape that can be used for filling is rectangles.

`?fillShapeLength` *`n_length`*

An optional argument that is a positive integer or floating-point number specifying the vertical measurement of the fill shape in user units. When it is not specified, it is considered equal to the value of *`n_width`*.

`?gap` *`S_gap`*

This optional argument is a character string that specifies the method used to fill the base shape. The only value supported is `minimum`.

`?justification` *`S_justification`*

This optional argument is a character string or symbol that specifies the origin of the first fill shape. It supports nine types of justification modes. These are: `lowerLeft`, `lowerCenter`, `lowerRight`, `centerLeft`, `centerCenter`, `centerRight`, `upperLeft`, `upperCenter`, and `upperRight`.

For the center configurations of the *`S_justification`* parameter, such as `lowerCenter`, `centerLeft`, `centerCenter`, `centerRight` and `upperCenter`, extra grid handling is done as follows:

For ring and c-type shapes, extra grid is adjusted inside the shape.

For shapes other than ring and c-type, extra grid is adjusted as below:

At right for `lowerCenter` and `upperCenter` configurations.

At the top for `centerLeft` and `centerRight` configurations.

At top right for `centerCenter` configuration.

Default Value: `lowerLeft`

`?keepOut` *`l_ptList`*

	Defines the list of polygon points that need to be excluded while filling. For example, (((x1, y1)(x2, y2) (x3, y3) (x4, y4))…)
`?spaceX` *`n_spaceX`*	
	Positive integer or floating-point number specifying the distance between the edges of fill shape in the direction of the X-axis.
`?spaceY` *`n_spaceY`*	
	An optional positive integer or floating-point number argument specifying the distance between the edges of fill shape in the direction of the Y axis. When it is not specified, it is considered equal to `n_spaceX`.
`?enclosureX` *`n_encX`*	
	Sets the minimum enclosure value for base shape and fill shape in the X-direction.
`?enclosureY` *`n_encY`*	
	This is an optional argument that sets the minimum enclosure value for base shape and fill shape in the Y-direction. When it is not specified, it is considered equal to *`n_encX`*.
`?grid` *`n_grid`*	
	This optional argument specifies the minimum grid resolution value. If not present, this value defaults to fill layer grid. If layer grid is not present, it defaults to manufacturing grid.
`?outputFillShape` *`l_dbId`*	
	This is an optional argument that can be either a valid list or valid database object `group_id`*.*

Value Returned

If value for `outputFillShape` is not specified, or if the value specified is `nil`, the value returned is:

`t`	Base shape is filled successfully
`nil`	Base shape could not be filled

If value of `outputFillShape` is a `groupid` (valid group ID in database), the value returned is:

d_group	Base shape is filled successfully with the shapes associated with the given group ID and the group of the filled IDs is returned.
`nil`	Base shape could not be filled

If value of `outputFillShape` is a `listShapes(valid list(nil) object)`, the value returned is:

l_shapes	Base shape is filled successfully with the given list and the list of filled IDs is returned.
`nil`	Base shape could not be filled

Examples

The following examples show the results of using `rodFillWithRects` in different circumstances:

Example 1: outputFilShape is specidfied as a valid list object

```
myList = list(nil)
returnList = rodFillWithRects(
        ?shapeId shapeId
        ?fillShapeLPP list("Cont" "drawing")
        ?fillShapeWidth 0.1
        ?fillShapeLength 0.1
        ?gap "minimum"
        ?justification "centerCenter"
        ?keepOut nil
        ?spaceX 0.1
        ?spaceY 0.1
        ?enclosureX 0.1
        ?enclosureY 0.1
        ?grid 0.005
        ?outputFillShape myList
)
```

Here:

- `returnList` contains list of filled shape IDs
- `myList` is the list of filled shape dbIds
- The return value `returnList` is same as `myList`, which is filled with shape IDs.

Example 2: outputFilShape is specidfied as a valid group object

```
returnGroup = rodFillWithRects(
        ?shapeId shapeId
        ?fillShapeLPP list("Cont" "drawing")
        ?fillShapeWidth 0.1
        ?fillShapeLength 0.1
        ?gap "minimum"
        ?justification "centerCenter"
        ?keepOut nil
        ?spaceX 0.1
        ?spaceY 0.1
        ?enclosureX 0.1
        ?enclosureY 0.1
        ?grid 0.005
        ?outputFillShape myGroup
)
```

Here,

- `myGroup` is a valid database object of group ID. This group includes a list of filled shape dbIds
- The value returned is `returnGroup`, which is same as `myGroup` that associates all filled shape IDs as members of this group.

Example 3: outputFilShape not given or given as nil

```
returnValue = rodFillWithRects(
        ?shapeId shapeId
        ?fillShapeLPP list("Cont" "drawing")
        ?fillShapeWidth 0.1
        ?fillShapeLength 0.1
        ?gap "minimum"
        ?justification "centerCenter"
        ?keepOut nil
        ?spaceX 0.1
        ?spaceY 0.1
        ?enclosureX 0.1
        ?enclosureY 0.1
        ?grid 0.005
)
```

The value returned is `t` if the function is successful in filling the base shape. The value returned is `nil` if the function fails in filling the base shape.

rodGetBBox

```
rodGetBBox(
     [ ?figId d_figId ]
     [ ?layer t_layer ]
     [ ?purpose t_purpose ]
     [ ?stopLevel x_stopLevel ]
     [ ?filterText g_filterText ]
     );end rodGetBBox
     => l_bBox / nil
```

Description

Returns the coordinates of the rectangular bounding box enclosing all the elements of a specified figure on a specified layer. You can optionally limit the calculation to include only elements on a specified layer purpose and down to a specified level of design hierarchy. You can also exclude labels and text displays from the calculation.

Arguments

`?figId` *`d_figId`*	Database ID of the figure for which the bounding box is required. A figure can be a shape, instance, via, or any type of figGroup (which can, in turn, contain shapes, instances, vias, and other figGroups). **Note:** Blockages and halos are not supported.
`?layer` *`t_layer`*	Name of the layer on which the function is to run; for example: `?layer "Poly"`
`?purpose` *`t_purpose`*	Name of the layer purpose on which the function is to run; for example: `?purpose "drawing"` If you do not specify a purpose, the function runs on all the purposes of the specified layer. Default: `nil` (all purposes)
`?stoplevel` *`x_stopLevel`*	Specifies how far down the hierarchy to search for elements to include in the calculation; for example: `?stopLevel 1`

The top level is level `0`. If you do not specify a stop level, the function traverses the entire hierarchy for the specified figure and layer.
Default: `32` (all levels)

`?filterText` *`g_filterText`*

Specifies whether labels and text displays are filtered out of the bounding box calculation. When set to `t`, labels and text displays are filtered out; when set to `nil`, labels and text displays are included.
Default: `nil`

Value Returned

`l_bBox`	List of coordinates defining the bounding box enclosing all the elements of the specified figure on the specified layer (with layer purpose, stop level, and text filters applied if specified). The bounding box is always defined by two sets of X and Y coordinates, the first specifying the lower left corner of the bounding box rectangle, the second specifying the upper right corner.
`nil`	No elements were found from which a bounding box could be calculated.

Example

The following example returns the bounding box of all the elements of the specified figure, taking into account shapes, instances, vias, and figGroups drawn on layer-purpose pair (`Poly drawing`) at the current level of hierarchy, but excluding labels and text displays.

```
rodGetBBox(?figId css() ?layer "Poly" ?purpose "drawing" ?stopLevel 1 ?filterText
t)
=> ((0.935 4.75) (6.28 7.43))
```

rodGetHandle

```
rodGetHandle(
     R_rodObj
     S_name
     );end rodGetHandle
     => g_handleValue / nil
```

Description

Returns the value of a system- or user-defined handle. You specify the handle name and the ROD object ID for the object with which the handle is associated.

For Boolean handles, the value can be either `t` or `nil`. When the value is `nil`, `rodGetHandle` returns an ambiguous result: `nil` could mean that the handle was not found or that the value of the handle is `nil`. To avoid ambiguity for Boolean handles, verify that the handle exists with the `rodIsHandle` function before you try to access it.

Arguments

`R_rodObj`	ROD object ID for the object with which the handle is associated.
`S_name`	Symbol or character string specifying the name of the handle whose value is returned. Enclose character strings in quotation marks (""); precede symbols with a single quotation mark ('). ROD converts symbols to character strings.

Value Returned

`g_handleValue`	The value of the handle. The values of point handles are fully transformed into the coordinate system of the top-level cellview.
`nil`	No handle was found, or, if the handle is Boolean, the handle value is equal to `nil`.

rodGetNamedShapes

```
rodGetNamedShapes(
     d_cvId
     );end rodGetNamedShapes
     => lR_rodObjID / nil
```

Description

Returns a list of the ROD object IDs for all named shapes (rectangles, polygons, and paths) at level zero in the hierarchy within the specified cellview that are ROD objects. This function looks only at the top level of hierarchy and does not return the ROD object ID for instances or cellviews.

Arguments

d_cvId	Database ID for the cellview in which you want to identify named shapes that are ROD objects. If the `rodGetNamedShapes` statement occurs in the body of a `pcDefinePCell` function or `tcCreateCDSDeviceClass` function call, the default value is `pcCellView` or `tcCellView`, respectively; otherwise, this argument is required. Default: none

Value Returned

lR_rodObjID	A list of the ROD object IDs for all named shapes in the specified cellview that are ROD objects.
`nil`	No ROD objects or other named shapes exist in the cellview, or an error occurred.

rodGetObj

```
rodGetObj(
     S_hierarchicalName | d_dbId
     [d_cellViewId]
     ); end rodGetObj
     => R_rodObj / nil
```

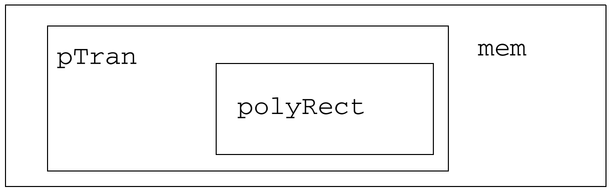
Description

Lets you find a named object at any level of hierarchy in your cellview. You must specify either the database ID for the object or its hierarchical name. If you specify the hierarchical name, you must also specify the cellview ID. When you specify a cellview ID, it is not necessary to also specify the database ID; however, if you specify both, the cellview ID must be correct for the database ID you specify. If your `rodGetObj` statement occurs in the code for a Pcell (in the body of a `pcDefinePCell` function or `tcCreateCDSDeviceClass` function call), specifying the cellview ID is optional; if you do not specify it, the cellview ID defaults to `pcCellView` or `tcCellView`, respectively.

Arguments

S_hierarchicalName	Symbol or character string specifying an object name, preceded by the names of the instances in the path to the object, separated by the slash character (/). If you specify this argument, you must specify the cellview ID as well, unless the statement is in the code for a Pcell or device. When you specify a cellview ID, it is not necessary to also specify the database ID; however, if you specify both, the cellview ID must be correct for the database ID you specify. Enclose character strings in quotation marks (""); precede symbols with a single quotation mark ('). ROD converts symbols to character strings. For this argument, an empty character string (null) is a valid value. For example, the hierarchical name for the shape `polyRect` shown below is `mem/pTran/polyRect`.

The hierarchical name for `polyRect` is `mem/pTran/polyRect`.



	An empty string ("") means that there is no named object; the function returns the ROD object ID for the cellview specified by *`d_cellViewId`*. Default: none.
`d_dbId`	Database ID for the object. If you specify this argument, you do not need to specify a cellview ID. However, if you specify both a database ID and a cellview ID, the cellview ID must be correct for the database ID you specify.
`d_cellViewId`	ID for the top-level cellview that contains the object specified by the *`S_hierarchicalName`* or *`d_dbId`* argument. When *`S_hierarchicalName`* is specified as an empty string (""), specify the cellview for which you want the ROD object ID. This argument is required when you do not specify *`d_dbId`*, unless the `rodGetObj` statement is in the code for a Pcell or device. If you specify this argument, you do not need to specify the database ID. However, if you specify both a database ID and a cellview ID, the cellview ID must be correct for the database ID you specify. Default: `pcCellView` or `tcCellView` when the `rodGetObj` statement occurs in the body of a `pcDefinePCell` function or `tcCreateDeviceClass` function call, respectively

Value Returned

`R_rodObj`	ROD object ID for the object.
`nil`	An error occurred and no object was found.

rodGetSubPart

```
rodGetSubPart(
     R_rodObjId
     t_subpartName
     );end rodGetSubPart
     => R_rodSubpartId / nil
```

Description

Returns the subpart ID corresponding to the specified subpart name for the multipart path (MPP) specified as `rodObjectId`.

Arguments

R_rodObjId	ROD object ID associated with the multipart path (MPP).ROD object ID associated with the multipart path (MPP).
t_subpartName	String specifying the subpart name for which the subpart ID is to be retrieved.

Value Returned

R_rodSubpartId	The subpart ID corresponding to the subpart name for the specified MPP.

Example

```
subId = rodGetSubPart(mppId "subpart0")
```

rodIsFigNameUnused

```
rodIsFigNameUnused(
     S_name
     [d_cvId]
     ); rodIsFigNameUnused
     => t / nil
```

Description

Determines whether the name specified by *`S_name`* is a valid name and whether it is already assigned to a ROD object, instance, or mosaic in the cellview specified by *`d_cvId`*. A valid name cannot contain hierarchy (indicated by one or more slashes) and cannot be an empty string.

You might want to verify whether a name is already assigned before you attempt to assign it to a shape with the rodNameShape function.

Arguments

`S_name`	Symbol or character string specifying the name you want to verify as valid and not assigned to a ROD object, instance, or mosaic in the specified cellview. Enclose character strings in quotation marks (""); precede symbols with a single quotation mark ('). ROD converts symbols to character strings. This argument is required For example, you can type the characters `testName` as `"testName"` or as `'testName`. Valid Values: any non-empty character string or symbol that does not contain a slash (/) Default: none
`d_cvId`	Database ID for the cellview in which you want to verify *`S_name`*. If the `rodIsFigNameUnused` statement occurs in the body of a `pcDefinePCell` function or `tcCreateDeviceClass` function call, the default value is `pcCellView` or `tcCellView`, respectively; otherwise, this argument is required. Default: none

Value Returned

`t`	The name is a valid name and is **not** assigned to a ROD object, instance, or mosaic in the cellview identified by *`d_cvId`*.
`nil`	The name is either already assigned to a ROD object, instance, or mosaic in the cellview identified by *`d_cvId`* or is not a valid name.

rodIsHandle

```
rodIsHandle(
     R_rodObj
     S_name
     );end rodIsHandle
     => t / nil
```

Description

Determines whether the name specified for *`S_name`* identifies a valid system- or user-defined handle associated with the object specified by *`R_rodObj`*. Use this function prior to the `rodGetHandle` function when you want to verify that the handle exists before you try to access it.

Arguments

`R_rodObj`	ROD object ID for the named object about which you are inquiring. This argument is required. Default: none
`S_name`	Symbol or character string specifying the name you want to verify as the name of a valid system- or user-defined handle for the object identified by *`R_rodObj`*. Enclose character strings in quotation marks (""); precede symbols with a single quotation mark ('). ROD converts symbols to character strings. This argument is required. Valid Values: any text string, enclosed in quotation marks Default: none

Value Returned

`t`	The name is a valid system- or user-defined handle for the object identified by *`R_rodObj`*.
`nil`	The name is not a valid system- or user-defined handle for the object identified by *`R_rodObj`*.

rodIsMasterChoppable

```
rodIsMasterChoppable(
     R_rodObjId
     )
     => t / nil
```

Description

Returns the value of the MPP master path choppability attribute.

Arguments

R_rodObjId	Specifies the master path of an MPP.

Value Returned

`t`	Indicates that the master path is choppable.
`nil`	Indicates that the master path is not choppable.

Example

In the following example, when the `choppable` attribute of master path is set to `nil`, the first subpart of the given MPP is made non-choppable. In this MPP, the `mppid` variable contains the rodObjId value.

```
unless(rodIsMasterChoppable(mppId)
     car(mppId~>subParts)~>choppable = nil
)
```

rodIsObj

```
rodIsObj(
     g_object
     ;end rodIsObj
     => t / nil
```

Description

Determines whether the object specified for `g_object` identifies a valid ROD object. This function is useful for testing variables and parameters to see if they contain a ROD object ID.

For information about what happens to ROD object IDs when you do an *Undo* in a cellview, see ROD object ID changes after Undo.

Arguments

`g_object`	Value of any data type. This argument is required. Default: none

Value Returned

`t`	The object is a valid ROD object.
`nil`	The object is not a valid ROD object.

Examples

The following examples show the results of using `rodIsObj` in several circumstances.

Example 1

Using `rodCreateRect` to create a rectangle results in a ROD object, and the function returns the ROD object ID. Assigning the result to the variable `rect` sets `rect` equal to the ROD object ID.

```
rect = rodCreateRect( ?cvId geGetEditCellView() ?layer "metal1" )
=> rodObj:19984408
```

Using `rodIsObj` to test `rect` returns `t`, as the object is a ROD object:

```
rodIsObj( rect )
t
```

Example 2

Testing anything other than a valid ROD object ID, such as a number, returns `nil`:

```
rodIsObj( 42 )
nil
```

Example 3

As in Example 1, above, the variable `rect` contains the ROD object ID for the new rectangle:

```
rect = rodCreateRect(?cvId cv
        ?name "r1"
        ?layer "metal1"
        ?width 5
        ?length 10
    ) ; end rodCreateRect
=> rodObj:23494680
```

Testing `rect` returns a ROD object ID:

```
rect
rodObj:23494680
```

Using `rodIsObj` to test `rect` returns `t`, as the object is a ROD object:

```
rodIsObj( rect )
t
```

However, if you select the ROD rectangle, move it, and then do an *Undo*, the variable `rect` no longer contains a ROD object ID.

```
rect
rod:invalid
```

```
rodIsObj(rect)
nil
```

For more information about what happens when you do an *Undo* in a cellview that contains ROD objects, see ROD object ID changes after Undo.

rodNameObject

```
rodNameObject(
     [ ?name S_name ]
     [ ?shapeId d_shapeId ]
     [ ?permitRename g_permitRename ]
     );end rodNameObject
     => R_rodObj / nil
```

Description

Assigns a name to an unnamed database shape or renames a ROD shape. Once the object is named, it can identified by a ROD object ID. This is an alias to an existing public function, rodNameShape.

Arguments

`?name` *`S_name`*

Symbol or character string specifying the name you want to assign to the shape. Enclose a character string in quotation marks (""); precede a symbol with a single quotation mark ('). ROD converts symbols to character strings. For example, to name an object `polyrect`, you can type the name as `"polyrect"` or as `'polyrect`. The name must be unique for the cellview.

The name cannot contain hierarchy, where hierarchy is indicated by one or more slashes. For example, at the lowest level in the hierarchy (zero-level), you can assign the name `polyRect`. You cannot assign a name with implied hierarchy, such as
`I1/I2/polyRect`.

When you do not specify a name, the system determines whether the shape is a rectangle, polygon, path, line, ellipse, donut, dot, label, and/or text-display object and assigns a name indicating the shape.

For rectangles, the system assigns `rect0` to the first rectangle you name in the cellview, `rect1` to the second, and so on.

For polygons, the system assigns `polygon0`, `polygon1`, and so on; for paths, `path`*`n`*.
For an ellipse, donut, dot, line, label, and/or text-display object, the system assigns `ellipse`*`n`*, `donut`*`n`*, `dot`*`n`*, `line`*`n`*, `label`*`n`*, or `text`*`n`*, respectively.
For a shape other than these, the system issues a warning message and the function fails.
Default: `rect`*`n`*, `polygon`*`n`*, `path`*`n`*, `ellipse`*`n`*, `via`*`n`*, `blockage`*`n`*, `boundary`*`n`*, `pathseg`*`n`*, `donut`*`n`*, `dot`*`n`*, `line`*`n`*, `label`*`n`*, or `text`*`n`*, as determined by the system.

`?shapeId` *`d_shapeId`*

Database ID of unnamed shape or database ID of a ROD object (not the ROD object ID; instances and mosaics are not shapes.)This argument is required.

`?permitRename` *`g_permitRename`*

A Boolean value specifying whether or not to rename the shape if it already has a name. The value must be `t` or `nil`.
Default: `nil`

Value Returned

`R_rodObj`	ROD object ID for ROD object information associated with the newly named shape.
`nil`	An error occurred and no shape was named.

Example

```
rodNameObject(
              ?name "myObj1"
              ?shapeId car(geGetSelSet())
             )
```

This will name the object selected as myObj1 and return the newly created ROD objectId for the object.

rodNameShape

```
rodNameShape(
     [ ?name S_name ]
     [ ?shapeId d_shapeId ]
     [ ?permitRename g_permitRename ]
     );end rodNameShape
     => R_rodObj / nil
```

Description

Assigns a name to an unnamed database shape or renames a ROD shape. Creates (or updates) a ROD object containing information associated with the shape, including its name and database ID. The associated ROD object is identified by a ROD object ID. Optionally, you can specify whether to rename an existing ROD object.

Note: Instances and mosaics are not shapes. You cannot rename an instance or a mosaic with this function.

You can refer to a named shape through hierarchy using its hierarchical name. You can also access system-defined handles associated with a named shape and create user-defined handles for a named shape.

To assign a name to an unnamed shape, you must identify the shape by its database ID. You can specify a name that is unique within the cellview or let the system generate a name. If you attempt to assign a name to an object that already has a name, and the *g_permitRename* argument is set to `nil`, the system displays an error message.

You can assign a property name and value, or list of property names and values, to the shape you are naming. For example, you could assign a Boolean property named `myProp` and a value of `t` or `nil` to the shape you are renaming.

Before assigning a name, you might want to verify that the string is a valid name and that it has not already been used in the cellview. To do this, use the rodIsFigNameUnused function.

Arguments

`?name` *S_name*

Symbol or character string specifying the name you want to assign to the shape. Enclose a character string in quotation marks (""); precede a symbol with a single quotation mark ('). ROD converts symbols to character strings.

For example, to name an object `polyrect`, you can type the name as `"polyrect"` or as `'polyrect`. The name must be unique for the cellview.

The name cannot contain hierarchy, where hierarchy is indicated by one or more slashes. For example, at the lowest level in the hierarchy (zero-level), you can assign the name `polyRect`. You cannot assign a name with implied hierarchy, such as
`I1/I2/polyRect`.

When you do not specify a name, the system determines whether the shape is a rectangle, polygon, path, line, ellipse, donut, dot, label, and/or text-display object and assigns a name indicating the shape. For rectangles, the system assigns `rect0` to the first rectangle you name in the cellview, `rect1` to the second, and so on. For polygons, the system assigns `polygon0`, `polygon1`, and so on; for paths, `path`*`n`*. For an ellipse, donut, dot, line, label, and/or text-display object, the system assigns `ellipse`*`n`*, `donut`*`n`*, `dot`*`n`*, `line`*`n`*, `label`*`n`*, or `text`*`n`*, respectively. For a shape other than these, the system issues a warning message and the function fails.
Default: `rect`*`n`*, `polygon`*`n`*, `path`*`n`*, `ellipse`*`n`*, `donut`*`n`*, `dot`*`n`*, `line`*`n`*, `label`*`n`*, or `text`*`n`*, as determined by the system.

`?shapeId` *`d_shapeId`*

Database ID of unnamed shape or database ID of a ROD object (not the ROD object ID; instances and mosaics are not shapes.)This argument is required.

`?permitRename` *`g_permitRename`*

A Boolean value specifying whether or not to rename the shape if it already has a name. The value must be `t` or `nil`.
Default: `nil`

Value Returned

`R_rodObj`	ROD object ID for ROD object information associated with the newly named shape.
`nil`	An error occurred and no shape was named.

rodPointX

```
rodPointX(
     l_point
     );end rodPointX
     => n_num / nil
```

Description

Returns the X coordinate of the point specified by *l_point*.

Arguments

l_point	A set of coordinates in one of the following formats: *x*:*y* or list(*x y*). Default: none

Value Returned

n_num	A signed integer or floating-point number that is the X coordinate of the point specified by *l_point*.
nil	No coordinate was returned due to an error.

rodPointY

```
rodPointY(
     l_point
     );end rodPointY
     => n_num / nil
```

Description

Returns the Y coordinate of the point specified by *`l_point`*.

Arguments

`l_point`	A set of coordinates in one of the following formats: *`x`*`:`*`y`* or `list(`*`x y`*`)`. Default: none

Value Returned

`n_num`	A signed integer or floating-point number that is the Y coordinate of the point specified by *`l_point`*.
`nil`	No coordinate was returned due to an error.

rodSetMasterChoppable

```
rodSetMasterChoppable(
     R_rodObjId
     g_setting { t | nil }
     )
     => t / nil
```

Description

Sets the choppability attribute of an MPP master path to the specified value.

Arguments

R_rodObjI	Specifies the master path of an MPP.
g_setting	The new setting for the choppability of the specified master path.

Value Returned

`t`	Indicates that the master path choppability attribute has been set to the given value.
`nil`	Indicates that either the given *`rodObjId`* is not a valid master path of an MPP or the master path choppability attribute could not be set to the given value (in this case, a warning message will be displayed).

Example

In the following example, the first subpart of an MPP created in the edited cellview, is made non-choppable.

```
cvId = geGetEditCellView()

mppId = rodCreatePath(
            ?layer          "metal1"
            ?pts            list(2:-15 2:-5 15:-5 15:-15)
            ?width          .8
            ?justification  "center"
            ?cvId cv
            ?offsetSubPath
            list(
                list(
                    ?layer          "vapox"
                    ?justification   "left"
                    ?sep             1
                    ?width           .4
                ) ;end of offset sublist1
                list(
                    ?layer           "vapox"
                    ?justification   "right"
                    ?sep             1
                    ?width           .4
                ) ;end of offset sublist2
            ) ;end of offset list of lists
        ) ;end of rodCreatePath

when(rodSetMasterChoppable(mppId nil)
     car(mppId~>subParts)~>choppable = nil
    )
```

rodSubPoints

```
rodSubPoints(
    l_point1
    l_point2
    ) ;end rodSubPoints
    => l_point / nil
```

Description

Subtracts *l_point2* from *l_point1* and returns the resulting point as *l_point*.

Arguments

l_point1	A set of coordinates in one of the following formats: *x*:*y* or `list(`*x y*`)` Default: none
l_point2	A set of points in one of the following formats: *x*:*y* or `list(`*x y*`)` Default: none

Value Returned

l_point	The point resulting from subtracting *l_point2* from *l_point1*.
`nil`	No point was subtracted due to an error.

rodUnAlign

```
rodUnAlign(
     R_rodObj1
     [R_rodObj2]
     );end rodUnAlign
     => t / nil
```

Description

When you specify only one ROD object ID, removes all alignments for the object specified. When you specify two ROD object IDs, removes only the alignment between the two objects.

Arguments

R_alignObj1	ROD object ID for the first or only object for which you want to remove alignments. When *R_alignObj2* is specified, only the alignments between the two objects are removed. When *R_alignObj2* is not specified, all alignments for *R_alignObj1* are removed. This argument is required. Default: none
R_alignObj2	ROD object ID for the second object for which you want to remove alignments. When specified, only the alignment between *R_alignObj2* and *R_alignObj1* are removed. Default: none

Value Returned

`t`	The alignment constraints were removed successfully.
`nil`	An error occurred and no alignment constraints were removed.

rodUnNameShape

```
rodUnNameShape(
     R1_rodObj
     );end rodUnNameShape
     => t / nil
```

Description

Removes the name from a named shape or list of named shapes. The ROD object ID for the shape and all user-defined handles associated with it are deleted. If the object is an instance, the instance keeps its name but all user-defined handles associated with the instance are deleted. Also removes all alignment constraints that reference the shape.

Arguments

R1_rodObj	ROD object ID or list of ROD object IDs that identifies the named shape for which you want to remove the name. Default: none

Value Returned

`t`	The name(s) of the shape(s) was removed and all alignment constraints that reference the shape(s) were deleted.
`nil`	An error occurred and no changes were made to the shape(s).

2

Relative Object Design Coordinate Functions

This chapter contains information on the Virtuoso relative object design (ROD) coordinate functions.

For more information on relative object design coordinate objects, refer to the *Virtuoso Relative Object Design User Guide*.

The functions in this chapter are listed in the alphabetical order.

rodCoordBisect

```
rodCoordBisect(
     o_rodCoord [ S_snapType ]
     );end rodCoordBisect
     => o_rodCoord
```

Description

Divides the input value by two (bisect). You can optionally set the argument that rounds off the calculated value, truncates it, or converts it based on the defined maximum ceiling.

Arguments

o_rodCoord	A `rodCoord` object.
S_snapType	A symbol representing snap type (ROUND, TRUNCATE, CEILING) ROUND– Converts the value to the nearest arithmetic unit with the mid-point value rounding away from the zero grid point. TRUNCATE – Converts the value to the next arithmetic unit towards the zero grid point. CEILING – Converts the value to the next larger arithmetic unit towards positive infinity. Default value: TRUNCATE.

Value Returned

o_rodCoord	Returns a new `rodCoord` object. A SKILL error is returned if the input value is invalid.

Example

```
rodCoordBisect(coord1)
  => rodCoord@0x016036
```

In this example, a new `rodCoord` object is returned.

rodCoordCreate

```
rodCoordCreate(
     g_inputValue [ S_unitType [ S_snapType ] ]
     [ x_resolution ]
     );end rodCoordCreate
     => o_rodCoordObj
```

Description

Creates a `rodCoord` object from the given value, that is, `g_inputValue`.

Arguments

g_inputValue	The input value for the constructor. It could be a float or integer number, or a string representing a float or integer number, or a `rodCoord` object. The `unitType` and `snapType` arguments are ignored when the input is a `rodCoord` object.
S_unitType	An optional argument that specifies a symbol representing the unit type of the input value, such as ARITHMETIC_UNIT or USER_UNIT. Default value: USER_UNIT.
S_snapType	A symbol representing the snap type (ROUND, TRUNCATE, CEILING). This argument is only applicable when the unit type is USER_UNIT. ROUND – Converts the value to the nearest arithmetic unit with the mid-point value rounding away from the zero grid point. TRUNCATE – Converts the value to the next arithmetic unit towards the zero grid point. CEILING – Converts the value to the next larger arithmetic unit towards positive infinity. Default value: ROUND.
x_resolution	An integer value that defines the number of arithmetic units in one user unit value.

Value Returned

o_rodCoord	Returns a new instance of the `rodCoord` user type. A SKILL error is returned if the input value is invalid.

Example

```
rodCoordCreate(1.25 'USER_UNIT 'ROUND)
 => rodCoord@0x018ex64
```

In this example, a new instance of the `rodCoord` object is returned.

rodCoordDefineGrid

```
rodCoordDefineGrid(
     n_gridValue
     x_resolutionValue
     );end rodCoordDefineGrid
     => t
```

Description

Defines the grid and the arithmetic resolution factor values, which are used by the `rodCoord*` API.

Arguments

n_gridValue	A fixed or floating number in the user unit that defines the grid value.
x_resolutionValue	An integer value that defines the number of arithmetic units in one user unit value.

Value Returned

t	The function is executed successfully.

Example

```
rodCoordDefineGrid(10.2 1000000)
=> t
```

In this example, the grid and arithmetic resolution factor values are defined.

rodCoordFix

```
rodCoordFix(
     o_rodCoord
     );rodCoordFix
     => o_newRodCoord
```

Description

Fixes the `rodCoord` object and returns a new `rodCoord` object using the `fix()` SKILL function.

Arguments

o_rodCoord	Specifies the `rodCoord` object.

Value Returned

o_newRodCoord	A new `rodCoord` object is returned. A SKILL error is returned if the input value is invalid.

Example

```
rodCoordFix(coord2)
=> rodCoord0x01604206
```

In this example, the `rodCoord` object, `coord2` is truncated and a new `rodCoord` object is returned.

rodCoordGetGrid

```
rodCoordGetGrid()
     ;end rodCoordGetGrid
     => o_gridValue
```

Description

Returns the grid value (defined by rodCoordDefineGrid) as a `rodCoord` object.

Arguments

None.

Value Returned

o_gridValue	A grid value defined by the rodCoordDefineGrid function as a `rodCoord` object is returned. A SKILL error is returned if the input value is invalid.

Example

```
rodCoordGetGrid()
=> rodCoord@0x018ef24
```

In this example, a grid value is returned as a `rodCoord` object.

rodCoordIsOnGrid

```
rodCoordIsOnGrid(
     o_rodCoord
     [ n_gridValue ]
     );end rodCoordIsOnGrid
     => t / nil
```

Description

Checks if the data in the given `rodCoord` object is on the grid or not.

Arguments

`o_rodCoord`	Specifies a valid `rodCoord` object.
`n_gridValue`	Specifies a fixed or floating number in user units that defines the grid value. If *`n_gridValue`* is not specified, the grid value (global) is set using the `rodCoordDefineGrid` function. However, if the global grid value is not set and the `rodCoordIsOnGrid` function is called without the second argument, the SKILL error is returned.

Value Returned

`t`	The data in the given `rodCoord` object is on the grid.
`nil`	The data in the given `rodCoord` object is not on the grid.

Example

```
rodCoordIsOnGrid(coord1)
=> t
```

In this example, the data in the specified `rodCoord` object is on the grid.

rodCoordParseString

```
rodCoordParseString(
     t_rodStringCoord
     )
     => o_rodCoord
```

Description

Creates a `rodCoord` object out of `t_rodString` generated by the `rodCoordToString` function.

Arguments

`t_rodStringCoord`	The string representation of the rodCoord object that is generated by the `rodCoordToString` SKILL function.

Value Returned

`o_rodCoord`	Returns a new `rodCoord` object. The error message is returned if the input value is invalid.

Example

```
; Call rodCoordToString with second argument t to generate a string in the needed
format:
rodCoordToString(coord t)
=> "1.22502@100000"
; Then pass this string to rodCoordParseString:
rodCoordParseString("1.22502@100000")
=> rodCoord@0x1b211400 ; rodCoord created (equal to the original rodcoord object).
```

rodCoordPartition

```
rodCoordPartition(
     o_dividend
     g_divisor
     );end rodCoordPartition
     => l_result
```

Description

Performs a division on the given arguments – dividend and divisor.

Arguments

o_dividend	Specifies a `rodCoord` object used as the dividend (numerator) for division.
g_divisor	Specifies a `rodCoord` object or a fixed number value used as the divisor (denominator) for division. If the divisor value is a fixed number it is used as a fixed number and not converted to `rodCoord` before applying the division. A zero value is not accepted.

Value Returned

l_result	A list containing two `rodCoord` objects – integer quotient and remainder.

Example

```
rodCoordPartition(coord1  coord2)
=> (rodCoord@0x018ef28  rodCoord@0x018ef68)
```

In this example, a division on the given arguments dividend and divisor is performed.

rodCoordSnap

```
rodCoordSnap(
     o_rodCoord
     [[ S_snapType ]
     [ n_gridValue ]]
     );end rodCoordSnap
     => o_rodCoord
```

Description

Snaps a `rodCoord` object so that its value is on the grid.

Arguments

o_rodCoord	A `rodCoord` object.
S_snapType	A symbol representing the snap type (ROUND, TRUNCATE, CEILING) ROUND – Converts the value to the nearest arithmetic unit with the mid-point value rounding away from the zero grid point. TRUNCATE – Converts the value to the next arithmetic unit towards the zero grid point. CEILING – Converts the value to the next larger arithmetic unit towards positive infinity. Default value: TRUNCATE.
n_gridValue	Specifies a fixed or floating number in user units that defines the grid value. If *n_gridValue* is not specified, the grid value (global) is set using the `rodCoordDefineGrid` function. However, if the global grid value is not set and the `rodCoordSnap` function is called without the third argument, the SKILL error is returned.

Value Returned

o_rodCoord	A new `rodCoord` object is returned. A SKILL error is returned if the input value is invalid.

Example

```
rodCoordSnap(coord1)
=> rodCoord@0x016036
```

In this example, a specified `o_rodCoord` object is snapped.

rodCoordToFloat

```
rodCoordToFloat(
     o_rodCoordObj
     );end rodCoordToFloat
     => f_value
```

Description

Returns the `rodCoord` data in user unit as floating point number.

Arguments

`o_rodCoordObj`	Specifies a valid `rodCoord` object.

Value Returned

`f_value`	A floating point number is returned. A SKILL error is returned if the input value is invalid.

Example

```
rodCoordToFloat(coord1)
=> 1.25
```

In this example, a floating point number is returned.

rodCoordToInt

```
rodCoordToInt(
     o_rodCoordObj
     );end rodCoordToInt
     => x_value
```

Description

Returns the `rodCoord` data in user unit as a fixed number.

Arguments

o_rodCoordObj	Specifies a valid `rodCoord` object.

Value Returned

f_value	A fixed number is returned. A SKILL error is returned if the input value is invalid.

Example

```
rodCoordToFloat(coord1)
=> 1
```

In this example, a fixed number is returned.

rodCoordToString

```
rodCoordToString(
     o_rodCoordObj
     [ g_dumpAR ]
     );end rodCoordToString
     => t_value
```

Description

Returns a string containing the `rodCoord` data in user unit as floating number.

Arguments

o_rodCoordObj	Specifies a valid `rodCoord` object.
g_dumpAR	If set to `t`, the string is dumped in the arithmetic resolution, `<number>@<ar>` format. You can then pass this string to the `rodCoordParseString` function to restore the `rodCoord` object.

Value Returned

f_value	A string value returned. A SKILL error is returned if the input value is invalid.

Example

```
rodCoordToFloat(coord1)
=> "1.25"
```

In this example, a string containing the `rodCoord` data in user unit as floating number is returned.

```
rodCoordToFloat(coord1 t)
=> "1.25@1000"
```

In this example, a string containing the `rodCoord` data is dumped as `1.25@1000`.