Virtuoso Technology Data ASCII Files Reference

Product Version ICADVM20.1
October 2020

Contents

Preface

This manual provides detailed information about the syntax of the statements you code in your technology and display resource files to define the constraints and display attributes for your design session.

This preface contains the following topics:

Scope

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

Label	Meaning
(ICADVM20.1 Only – 95512)	Features supported only in the ICADVM20.1 release and which require the Virtuoso_Adv_Node_Opt_Lay_Std license (95512).
(ICADVM20.1 Only – 95511)	Features supported only in the ICADVM20.1 release and which require the Virtuoso_Adv_Node_Opt_Layout license (95511).

(ICADVM20.1 Only – 95511 and 95800)	Features supported only in the ICADVM20.1 release and which require the Virtuoso_Adv_Node_Opt_Layout (95511) and Virtuoso_Layout_Suite_EXL (95800) licenses.
(ICADVM20.1 Only – Virtuoso MultiTech Framework)	Features supported only in the ICADVM20.1 release and which require the Virtuoso_MultiTech_Framework (95022) license.
(ICADVM20.1 Only – Photonics)	Features supported only in the ICADVM20.1 release and which require the Virtuoso_Photonics_Option license (95550).
(IC6.1.8 Only)	Features supported only in the IC6.1.8 release.

Licensing Requirements

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

Related Documentation

What's New and KPNS

- *Virtuoso Technology Data What's New*
- *Virtuoso Technology Data 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 ASCII Files Reference*
- *Virtuoso Technology Data Constraints Reference*
- *Virtuoso Technology Data User Guide*
- *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 Multi-Patterning Technology User Guide*
- *Virtuoso Placer User Guide*
- *Virtuoso Simulation Driven Interactive Routing User Guide*
- *Virtuoso Width Spacing Patterns User Guide*
- *Virtuoso RF Solution Guide*
- *Virtuoso Electromagnetic Solver Assistant 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 Relative Object Design SKILL Reference*
- *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.

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.

Note: All values are specified in user units unless otherwise noted.

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
r	defstruct	defstruct
R	rodObj	relative object design (ROD) object

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

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

1 The ASCII Technology File

An ASCII technology file defines the contents of a technology database. This chapter contains high-level information about a technology file. It contains the following topics:

- Introduction to the ASCII Technology File
- Technology File Structure
- Syntactical Guidelines for an ASCII Technology File

Introduction to the ASCII Technology File

An ASCII technology file defines the contents of a technology database. Specifying the data for incremental technology databases in multiple technology files and compiling the ASCII technology files in their reference order creates an incremental technology database graph. This guide details how to specify data in an ASCII technology file.

Each ASCII technology file is made up of sections and subsections that must be specified according to the structure and syntax. The various sections of the technology file allow you to specify and define technology file controls; design layers, layer attributes, sites, vias, and devices; design constraints; and the layer display in the Palette.

Important

When you specify a constraint, you must use its proper name, as defined in the *Virtuoso Technology Data Constraints Reference*, and specify it according to the ASCII syntax defined in this manual. Conforming to these guidelines allows interoperability between tools, which means that the constraints can be interpreted by any applications that use them.

To provide the greatest flexibility and satisfy the needs of a wide range of applications, the technology file provides a place to define many different constraints and rules. Applications can require that you define specific technology data and constraints in your technology file. Refer to the documentation for any application you are using to determine the technology file requirements and to ensure that your technology data is complete and is used by the application. User-defined technology data can be included in the technology file, but is not recognized or used by Cadence applications.

The technology file also allows you to set up multiple constraint groups that define design constraints, which can then be applied during design sessions based on the applications in use. Constraint groups allow a flexibility of application that is not characteristic of design rules specified in other parts of the technology file. Refer to the application documentation to determine the constraint groups it recognizes and the order of precedence it applies to them.

A technology file must be compiled with the Technology Tool Box *New* or *Load* command before it can be used. ASCII technology file functions cannot be typed directly into a shell window or the Command Interpreter Window (CIW); they can be specified only in an ASCII technology file. The *Virtuoso Technology Data User Guide* contains detailed information about setting up and creating incremental technology databases, including information important for specifying ASCII technology files, such as information about planning technology database contents and about the technology data you can and cannot duplicate in multiple technology databases in a graph.

Note: Another important file in this context is a display resource file. It defines the packets that the design software uses to display the layers in your design. It also defines the different display devices, such as monitors and plotters, that you use. For display resource file syntax, see Display Resource File.

Related Topics

- Technology File Structure
- Syntactical Guidelines for an ASCII Technology File

Technology File Structure

The technology file contains the following statements and sections:

Technology file section	Establishes	Technology file subsection
Statement: include	A file to be included in the technology file	N/A
Statement: comment	A comment to be preserved during technology file loading and dumping	N/A
controls	A process node setting for the technology database	processNode
	Technology file version	techVersion
	Technology parameters for use throughout the technology file	techParams
	Distance measure type: Euclidian or Manhattan	distanceMeasure
	Units for view types	viewTypeUnits
	Overall design manufacturing grid resolution	mfgGridResolution
	An ordered list of reference technology databases	refTechLibs
	A process family name for the technology database	processFamily
	The fabric type of the technology database	fabricType

Technology file section	Establishes	Technology file subsection
layerDefinitions	Layers	techLayers
	Layer purposes	techPurposes
	Priorities for layer-purpose pairs	techLayerPurposePriorities
	Displays	techDisplays
	Properties on layers	techLayerProperties
	Derived layers	techDerivedLayers
layerRules	Equivalent layers	incompatibleLayers
	Layer functions	functions
	Layer attributes	analysisAttributes
	Layer manufacturing resolutions	mfgResolutions
	Layer routing directions	routingDirections
	Electrical characterization rules	Specified as `currentDensity` or `currentDensityTables` rules: ■ peakACCurrentDensity ■ avgACCurrentDensity ■ rmsACCurrentDensity ■ avgDCCurrentDensity
constraintGroups	Constraint groups	N/A
siteDefs	Scalar site definitions	scalarSiteDefs
	Array site definitions	arraySiteDefs
viaDefs	Standard via definitions	standardViaDefs
	Custom via definitions	customViaDefs
viaSpecs	Via specifications	N/A

Technology file section	Establishes	Technology file subsection
devices	(Activates) predefined Cadence device classes, declares devices, establishes multipart path templates, and defined templates for waveguide objects	tcCreateCDSDeviceClass ruleContactDevice multipartPathTemplates waveguideTemplates
leRules	Layers displayed initially in Palette	leLswLayers

Related Topics

- Introduction to the ASCII Technology File
- Syntactical Guidelines for an ASCII Technology File

Syntactical Guidelines for an ASCII Technology File

Important

See *Virtuoso Technology Data User Guide* for information about properly planning and defining data to fit into an incremental technology database graph. This manual focuses on the data you can specify in an individual ASCII technology file, without consideration for setting one up for an incremental technology database.

Consider the following guidelines for specifying technology file data in an ASCII technology file:

- You must supply all arguments unless they are identified as optional by being shown in square brackets (`[]`) in the syntax specification.
- You must specify at least one argument, but can specify more, if the argument is shown followed by an ellipsis (…) in the syntax specification in related topics.
- You must specify keywords precisely as shown.
- You must specify constraints precisely as shown and intend them to be applied as described.
- You must specify the appropriate data type for arguments.

  For more information, see Identifiers Used to Denote Data Types.
- You can specify an expression for any user-defined argument.
- For any *`layer`* value, you can specify either a layer name or a layer number.

Note: Do not apply properties or attributes to system-reserved layers or purposes. The software discards any user customization applied to reserved layers or purposes.

Related Topics

- Introduction to the ASCII Technology File
- Technology File Structure

2

Technology File Statements

Statements let you include in the technology file other technology files and comments. They provide flexibility in technology file development and maintenance, rather than provide design data.

This chapter contains the following topics:

- include
- comment

include

```
include( "t_techFileName" )
```

where, `t_techFileName` is the name of the technology file to include.

Description

Includes another file defining technology data with the main technology file. For example, assume that your technology data contains extensive device definitions. For ease of file maintenance, you might want to create a separate file containing the devices data for your technology file. The `include` statement enables you to put this data in a separate technology file and reference that file from your main technology file.

Example

```
include("/usr1/smith/devices.def")
```

This statement, placed in the technology file where the `devices` section belongs, includes in the technology file the file `devices.def` from the location `/usr1/smith`. When the compiler encounters this statement, it retrieves and compiles the `devices.def` file as part of the current technology library.

Note: If you compile a technology file containing an `include` statement and then dump the ASCII technology file from the resultant technology library, the dumped technology file will not contain the `include` statement but will contain the included technology file data instead.

comment

`comment ( "`*`t_comment`*`" )`

where, *`t_comment`* is the comment text, which must be enclosed in quotation marks.

Description

Lets you add comments that are preserved throughout compilation and subsequent technology file dumping. Comment statements must be added at the section level; they cannot be added within another section. A comment statement applies to the section that immediately follows it.

You can also add comments by preceding them with a semicolon (;). However, comments added in this way are not preserved when you compile a technology file into a technology library and later dump an ASCII technology file from a technology library.

Example

```
comment(
    "This comment applies to the controls section."
)
controls(

    techParams (
        (theta   2.0)
        (lambda  4.0)
    )
)
```

When you compile the technology file containing these statements into a technology library, the comment is assigned to the section immediately below, in this case, the `controls` section. If you subsequently dump the technology data from this library to an ASCII file, the comment is preserved along with the `controls` section data. Other comments, identified by semicolons, are not preserved.

3

Technology File Controls

Technology file controls let you define data that can be used throughout the technology file.

This chapter contains the following topics:

controls

```
controls()
```

Description

When specified, `controls` must be the first section in the technology file. Subsections that specify specific controls must be enclosed in this section.

Additionally, the `controls` section must be loaded only once, that is, a technology file must contain only one `controls` section. If multiple technology files are loaded by using the `include` statement, the `controls` section must be present only in the main technology file that refers to other technology files.

processNode

```
controls(
     processNode( g_processNodeValue )

)
```

Description

Associates a process node setting with the technology database. The value is specified in user units, using an integer or a floating-point number. It is converted to database units by using the `dbuPerUU` from the `maskLayout` viewType.

The technology file compiler is updated to compile the new syntax. If conflicts are detected, compilation fails. The technology file dumper updates to dump out the new (local) attribute when set on a technology.

Arguments

g_processNodeValue	The value is specified in user units, using an integer or a floating-point number. It is illegal to specify the value as `0`.

Example

```
controls(
    processNode(0.02)
)
```

The value 0.02 is associated with the process node.

Related Topics

- controls

techVersion

```
controls(
     techVersion( s_version )

)
```

Description

Specifies the technology file version.

Introduction to techVersion 1.0

Virtuoso Version	techVersion	Behavioral Changes
IC_6.1.5	1.0	`cdsViaDevice` is always created as a cdsVia. All entries in constraint tables need to be specified. If you have more than one table-based constraint per constraint definition and layer(s) in the technology file, techVersion must be specified as 1.0.

Arguments

`s_version`	The currently supported technology file syntax version is "`1.0`".

Caution

To compile a technology file compatible with the technology file syntax of earlier versions of Virtuoso, do not use `techVersion()`***.***

Example

```
controls(
    techVersion("1.0")
)
```

The only allowed value is `1.0`. The cdsVias are retained only if `techVersion("1.0")` is specified. If there is no `techVersion` section, it symbolizes a legacy IC6.1.4 ASCII technology file.

Related Topics

- controls

techParams

```
controls(
     techParams(
     ( t_name g_value )
     …
     )
)
```

Description

Defines parameters that can be used throughout the technology file. Once you have defined a parameter with `techParams`, you can invoke it wherever needed in your technology file, with the exception of table indexes, by specifying it with `techParam(` *`paramName`* `)`.

Note: Parameters cannot be specified as table indexes. For more information, see Current Density Tables and Spacing Table Constraint.

When you use the `techParam()` function in any section of the technology file, the technology file compiler makes an explicit reference to the `controls` section, instead of evaluating the expression and storing only the value. As a result, if, at a later date, you want to update the technology file functions that use a control parameter, you need to update the value of the parameter at one place.

When you dump the technology file, the control parameters, rather than the evaluated expressions, appear in the ASCII syntax. You can reverse this behavior with the `techSetEvaluate()` SKILL function.

Arguments

t_name	The name of the parameter.
g_value	The value to assign to the parameter.

Example

```
techParams(
 ; (name    value)
   (lambda  0.3  )
)
```

This example assigns a value of `0.3` to the parameter `lambda`. When you use the expression `techParam("lambda")` as a value in any technology file specification, the system uses the value stored for the parameter in this section. For example, the following specification sets the `minSpacing` constraint on `Metal1` to `0.3`.

```
spacings(minSpacing "Metal1" techParam("lambda"))
```

Related Topics

controls

distanceMeasure

```
controls(
     distanceMeasure(
          ( 'euclidian | 'manhattan )
     )
     …

)
```

Description

Specifies Euclidian or Manhattan spacing as the default for all constraints in the technology file, without the selection being set explicitly for individual constraints. The default is `'euclidian`.

Related Topics

controls

viewTypeUnits

```
controls(
     viewTypeUnits(
          ( t_viewType t_userUnit n_dbuPerUU )
          …
     )
)
```

Description

Specifies the user units and database units per user unit for the specified view type. To apply a `dbuPerUU` value other than the default of 1000, this subsection must be specified before the mfgGridResolution subsection in the technology file.

Arguments

`t_viewType`	The view type, for example, `maskLayout`, `schematic`, `schematicSymbol`, `netlist`, and `hierDesign`.
`t_userUnit`	The user unit to be used for the view type, for example, `"micron"`, `"inch"`.
`n_dbuPerUU`	The number of database units per user unit. Default: `1000`

Example

```
viewTypeUnits(
    ( maskLayout          "micron"        2000 )
    ( schematic           "inch"          160  )
    ( schematicSymbol     "inch"          160  )
    ( netlist             "inch"          160  )
    ( hierDesign          "_def_"         2000 )
) ;viewTypeUnits
```

Note: TechDB enforces that `hierDesign` type `dbuperUU` is the same as the `maskLayout` type `dbuPerUU` in the techDB reference graph. The same policy applies to the `userUnit` attributes of `maskLayout` and `hierDesign` in the techDB reference graph.

Caution

The information stored in hierDesign consists of constraints that are usually derived from techDB. In addition, techDB itself retrieves its dbuPerUU from the maskLayout viewType. Therefore, in a scenario where hierDesign view constraint groups refer to rules in a techDB that is at 2000 dbuPerUU because it inherits from a maskLayout that is at 2000 dbuPerUU, it would generate an error if you stored items at 1000 dbuPerUU.

Related Topics

controls

mfgGridResolution

```
controls(
     mfgGridResolution(
          (g_value)
     )
)
```

Description

Specifies the manufacturing grid resolution; grid snapping must be a multiple of the specified value. The specified value applies to the entire design, except to specific layers that are assigned their own routing grid resolutions by the mfgResolutions constraint. This subsection must be placed after any viewTypeUnits specification in the technology file.

Arguments

g_value	The manufacturing grid resolution.

Example

```
mfgGridResolution(
    (0.001000)
) ;mfgGridResolution
```

Sets the manufacturing grid resolution to `0.001000`.

Related Topics

controls

refTechLibs

```
controls(
     refTechLibs(t_techLibName …)
)
```

Description

Specifies references to other technology libraries from the technology library defined by this ASCII technology file. The library list is an ordered list indicating the order of the referenced technology libraries in an incremental technology database graph.

For information about setting up ASCII technology files for incremental technology databases, see *Virtuoso Technology Data User Guide*.

Arguments

t_techLibName	The name of the reference library.

Example

```
refTechLibs("HVStdCells" "6MP&R" "90Core")
```

Specifies the technology libraries referenced by the technology library created by this ASCII technology file: `HVStdCells`, then `6MP&R`, then `90Core`.

Related Topics

controls

processFamily

```
controls(
     processFamily( t_processFamilyName )
)
```

Description

Associates a process family name with the technology database. This allows identification of technology databases that belong in the same graph, or, conversely, technology databases with different names that do not belong in the same graph. Technology databases without a process family specified can be used with any other technology database in a graph.

Arguments

t_processFamilyName	The name of the process family. Valid values: A string, an empty string, or a string containing a space

Note: Compiling a technology file with the process family specified as an empty string or a string containing a space in "replace" mode unsets the process family specified in the technology database; compiling in "merge" mode makes no change in the original process family.

Example

```
processFamily("CMOS90")
```

Specifies that the technology database belongs to the `CMOS90` process family, which is used, in this example, for `CMOS90` design.

Related Topics

controls

fabricType

```
controls(
     fabricType( t_fabricType )
)
```

Description

Specifies the fabric type of a technology database.

Arguments

`t_fabricType`	The fabric type of the technology database. The default value is `unspecified`. The other supported values are `ic`, `package`, `board`, and `module`.
	Note: You need the `Virtuoso_MultiTech_Framework` license to use a fabric type other than `ic`. You can change the fabric type to `ic` without this license if the technology database does not have any package elements, such as wirebond profiles.
	After a fabric type is specified for one technology database, all other technology databases in the technology graph it is in must have the same fabric type or the value `unspecified`.
	The following values correspond to Cadence package names: ■ `package`: Cadence® SiP Layout `.sip` file ■ `module`: Cadence® Allegro® Package Designer `.mcm` file ■ `board`: Cadence® Allegro® PCB Designer `.brd` file

Example

```
fabricType("package")
```

Specifies that the technology database has the fabric type `package`.

Related Topics

controls

4

Technology File Layer Definitions

The `layerDefinitions` section of the technology file specifies the user-defined layers that can be used throughout the technology databases in a graph and in design sessions.

If a layer or purpose name contains a special character, it must be enclosed in double quotes. Names that do not contain special characters may or may not be enclosed in double quotes. For a list of special characters, see the section on "Special Characters" in *Cadence SKILL Language User Guide*.

Note: Do not redefine system-reserved layer or purpose names in any way. The software discards any such customization. For limitations that other Cadence or third-party tools might have with regard to layer and purpose names, refer the relevant documentation.

Important

Any object reference in the ASCII technology file must first be defined in the technology database.

This chapter contains the following topics:

layerDefinitions

```
layerDefinitions()
```

Description

Contains layer definitions. All subsections specifying layer definitions must be enclosed within the parentheses of this section. The subsections are:

techLayers

```
techLayers(
     ( t_layerName x_layerNumber t_layerAbbr [x_maskNumber] ['valid t/nil]
     ['allowSetToValid t/nil] )
     …
) ;techLayers
```

Description

Defines the layers to be used throughout the technology file.

Note: Applications that display layer names do not always have the room to display the entire layer name. The optional abbreviation expands your control over what is displayed in narrow fields. An application can display one or more of the following values:

- Layer name
- Layer name truncated to fit or the layer name abbreviation

Arguments

t_layerName	Name of the layer. Valid values: Any string; layer names must be unique
x_layerNumber	Number of the layer. Valid values: A unique integer from `0` through `194` and from `256` through `2`^{31}`-1`. **Note:** Layers numbered `195` through `255` are system-reserved layers.
t_layerAbbr	A short string that is an abbreviation for the layer name.
x_maskNumber	The number of masks that can be applied to a layer. **Note:** By default, there is one mask per layer. For multiple patterning, you must specify the number of masks for layers that are MPT-enabled.
`'valid t/nil`	The default value of this boolean attribute is `nil`. The dumper dumps this attribute only if it is set to `t`.
`'allowSetToValid t/nil`	The default value of this boolean attribute is `nil`. The dumper dumps this attribute only if it is set to `t`.

Example

```
techLayers(
 ; ( layerName      layerNumber     abbreviation   'allowSetToValid t/nil)
   ( nimplant       39              nimplnt        'allowSetToValid nil)
)
```

techPurposes

```
techPurposes(
     ( t_purposeName x_purposeNum [t_purposeAbbr]
       ['parent tx_parentPurpose] ['voltageRange (f_min f_max)]
       ['sigType t_sigType] ['description t_description] ['valid t/nil]
       ['allowSetToValid t/nil]
     )
) ;techPurposes
```

Description

Defines purposes that can be used to define layer-purpose pairs. Layer-purpose pairs are used for display during design sessions.

Note: Applications that display purpose names do not always have the room to display the full purpose name. The optional abbreviation extends your control over what is displayed in the fields that are too narrow to display the full purpose name. An application can display one or more of the following values:

- Purpose name
- Purpose name truncated to fit or the purpose name abbreviation
- Parent purpose name or number
- Voltage range
- Signal type
- Purpose description

Arguments

t_purposeName	The name of the purpose. Valid values: Any string that is not a system-reserved purpose.
x_purposeNum	The purpose number. Valid values: A unique integer, `1` through `128` and `256` through 2^{32}`-65535`. **Note:** Purpose numbers `129` through `255` are system-reserved. Purpose number `0` is also system-reserved and is defined as "unknown".

`t_purposeAbbr`	The abbreviation for the purpose name. Valid values: Any string of seven characters or less.
`'parent x_parentPurpose`	The name or number of the parent purpose. The default parent purpose is `drawing`. There is a restriction on the use of `drawing` with `'sigType`. For more information, see the description below for `'sigType`. Valid values: Predefined purposes. For more information, see Predefined Purposes.
`'voltageRange f_min f_max`	The minimum and maximum voltage pair. It specifies the voltage range carried by the shapes created with the specified purpose.
`'sigType t_sigType`	This string attribute does not have a default value and it cannot be an empty string. If *`sigType`* is `signal` or a user-defined value, any purpose can be specified as a parent purpose. If *`sigType`* is a built-in signal type other than `signal`, such as `power` or `ground`, the parent purpose must be `drawing`.
`'description t_description`	This string attribute does not have a default value. It is used to specify a brief description of the purpose. It cannot be an empty string.
`'valid t/nil`	The default value of this Boolean attribute is `nil`. The dumper dumps only this attribute if it is set to `t`.
`'allowSetToValid t/nil`	The default value of this Boolean attribute is `nil`. The dumper dumps only this attribute if it is set to `t`.

Predefined Purposes

The purpose names listed in the table below are predefined (in `cdsDefTechLib`) and must be used in accordance with their defined meaning. Whenever possible, these purposes should be used instead of user-defined purpose names.

Purpose	**Description**
`drawing`	Is the default purpose used for shapes that carry signals
`fill`	Identifies automatically generated fill shapes, which may be removed automatically by place and route tools, as needed. These do not carry signals and are generated and added to enable the design to pass density checks. These must be uniquely identified so that they can be removed to make room to move a route, if needed. Fill shapes are not subject to the same rules as regular drawing shapes and need not be completely correct because they do not carry a signal. For example, sometimes the rules for fill shapes allow them to be closer together than drawing shapes are allowed to be.
`slot`	Identifies shapes for slotting modification during the manufacturing process
`OPCSerif`	Identifies the data created by optical proximity correction
`OPCAntiSerif`	Identifies the data created by optical proximity correction
`annotation`	Used for chip annotation, such as for labels and logos
`gapFill`	Identifies fill shapes added to correct DRC errors such as notch, min-step, and enclosed-area violations
`redundant`	Identifies objects that have been placed more than once, such as double vias on a route
`fillOPC`	Identifies automatically generated fill shapes that might have shapes added to them by Optical Proximity Correction (OPC) tools. For example, very small shapes might be added to the corners during OPC when run as part of mask preparation to ensure that the corners are printed correctly on the chip. These shapes typically follow the same rules as regular drawing shapes but like with a regular fill shape, a router can remove these to modify a route because they can be regenerated.

Purpose	Description
`customFill`	Identifies fill shapes that are hand-drawn or generated as part of a Pcell and added manually by a Virtuoso user in a custom design flow
`fatal`	Used for fatal error markers
`critical`	Used for critical error markers
`soCritical`	Used for signed-off critical error markers
`soError`	Used for signed-off error markers
`ackWarn`	Used for acknowledged warning markers
`info`	Used for information markers
`track`	Used for track patterns
`blockage`	Identifies objects associated with blockages
`grid`	Used for grids such as placement and manufacturing grids
`warning`	Used for warning markers
`tool1`	Identifies shapes that are created when a DRC check is run
`tool0`	Identifies shapes that are created when a DRC check is run
`label`	Used for labels, such as pin, instance, device, and PR boundary labels
`flight`	Used for flight lines
`error`	Used for error markers
`annotate`	Used for user-generated information
`drawing1`, ..., `drawing9`	Used for objects such as congestion maps (designFlow), highlights, annotations, grids, and devices
`boundary`	Used for delineating objects such as groups and rows and for PR, snap, and cluster boundaries
`pin`	Identifies objects associated with pins
`net`	Identifies objects associated with nets
`cell`	Identifies an object that is a block
`all`	Symbolizes that an object is not purpose-specific and applies to all purposes of a layer

Customizing Object Attributes

You can control how shapes are displayed on various layers by defining layer-purpose pairs and assigning to each layer-purpose pair a display packet. If you want to prevent shapes from being drawn on a layer-purpose pair, you can set the `valid` attribute for that layer-purpose pair to `nil`.

Various combinations of the layers and purposes specified in `cdsDefTechLib` define a set of system-reserved layer-purpose pairs that are used to display system-generated output, such as markers, rulers, flight lines, and boundaries. None of the system-reserved layer-purpose pairs are set to valid by default. As a result, they are not listed automatically in the Palette. For more information about the Palette, see Palette Assistant in Virtuoso Layout Suite documentation.

Note: Shapes can be drawn on all layer-purpose pairs due to CDB compatibility.

Layout Objects

The table below lists layout objects and the associated LPPs.

Object	**Layer**	**Purpose**	**Packet**
Guides, steiners	`annotate`	`drawing`	`annotate`
Instances, vias, mosaics (BBox)	`instance`	`drawing`	`instance`
Markers	`marker`	`warning`	`markerWarn`
		`error`	`markerErr`
		`annotate`	`markerAno`
		`info`	`markerInf`
		`ackWarn`	`markerAck`
		`soError`	`markerSer`
		`soCritical`	`markerScr`
		`critical`	`markerCrt`
		`fatal`	`markerFat`
Rows	`Row`	`boundary`	`RowBnd`
Blockages	Any physical layer	`blockage`	*View in DRE**
Placement blockages	`Cannotoccupy`	`Boundary`	`CannotoccupyBnd`

Object	**Layer**	**Purpose**	**Packet**
PR boundary, blockages	`prBoundary`	`boundary`	`prBoundaryBnd`
Area boundary, figGroups, custom rows	`border`	`boundary`	`area`
Cluster boundary	`Group`	`boundary`	`GroupBnd`
Snap boundary	`snap`	`boundary`	`snap`
Track pattern	Any physical layer	`track`	*View in DRE*[*]
Manufacturing grid	Any physical layer	`grid`	*View in DRE*[*]
Placement grid	`snap`	`grid`	`snap`
Rulers	`marker`	`annotate`	`markerAno`
	`hilite`	`drawing4`	`hilite4`
Symmetric axes	`edgeLayer`	`drawing`	`edgeLayer`
Selection	`hilite`	`drawing`	`hilite`
Highlight (command-specific)	`hilite`	`drawing1-drawing9`	`hilite1-hilite9`
Instance BBox (when edit-in-place is active and you point to an instance)	`hilite`	`drawing1`	`hilite1`
Instance and figGroup BBox (when being edited in place)	`hilite`	`drawing2`	`hilite2`
Reference point	`hilite`	`drawing3`	`hilite3`

*For information about how to view the complete list of associated display packets in Display Resource Editor (DRE), see Using Display Resource Editor in *Virtuoso® Technology Data User Guide*.

Analog Objects

The table below lists analog objects and the associated LPPs.

Object	**Layer**	**Purpose**	**Packet**
Analog instance label (`cdsName()`)	`annotate`	`drawing7`	`annotate7`
Analog pin annotation (`cdsTerm()`)	`annotate`	`drawing8`	`annotate8`
Analog device annotation (`cdsParam()`)	`annotate`	`drawing`	`annotate`
Pin annotate (pin net expressions)	`pin`	`annotate`	`pinAnt`
Waveforms **Note:** The display settings in the `display.drf` files are not considered while running sweeps, corners, and parametric analysis in ADE. For more information, see `useDisplayDrf`.	`y0-y9`	`drawing`	`y0-y9`
Circuit analysis boundary annotation	`annotate`	`boundary`	`AnnotateBoundary`

Schematic Objects

For information about schematic objects and their associated LPPs, see Customizing Schematic Object Attributes in *Virtuoso® Schematic Editor L User Guide*.

Dynamic Highlight and Drag Outline

The display packets listed below are used for displaying the dynamic highlight and the outline of an object being dragged or moved.

Object	**Packet**
Dynamic highlight For more information, see Setting Dynamic Highlights in *Virtuoso® Technology Data User Guide*.	`useDynamicHilightPacket`
Drag outline	`monoColorDragPacket`

Example 1

```
techPurposes(
    (highvoltage  13  hvo  'sigType "analog"  'voltageRange (0.0  0.3)
    'description "highvoltage range 0.0 --- 0.3."
    )
)
```

Example 2

```
techPurposes(
    (lowvoltage  14  lvo  'parent slot  'sigType "signal"
    'voltageRange (-3.3 0.0) 'description "lowvoltage range -3.3 --- 0.0"
    )
)
```

Example 3

```
techPurposes(
    (techPurpose2  2  tp2  'parent fill  'sigType "userDef2"
    'description "This purpose is for user-defined signal type 2."
    )

)
```

Example 4

```
techPurposes(
    (blockage1  1214  BLK1  'parent annotation  'valid nil )
)
```

Example 5

```
techPurposes(
    (blockage1  1214  BLK1  'parent annotation  'valid nil  'allowSetToValid nil)
)
```

techLayerPurposePriorities

```
techLayerPurposePriorities(
     ( t_layerName t_purposeName )
     …
) ;techLayerPurposePriorities
```

Description

Lists layers with assigned purposes in the order of priority, from the lowest to the highest.

Arguments

t_layerName	Name of the layer.
t_purposeName	Name of the purpose.

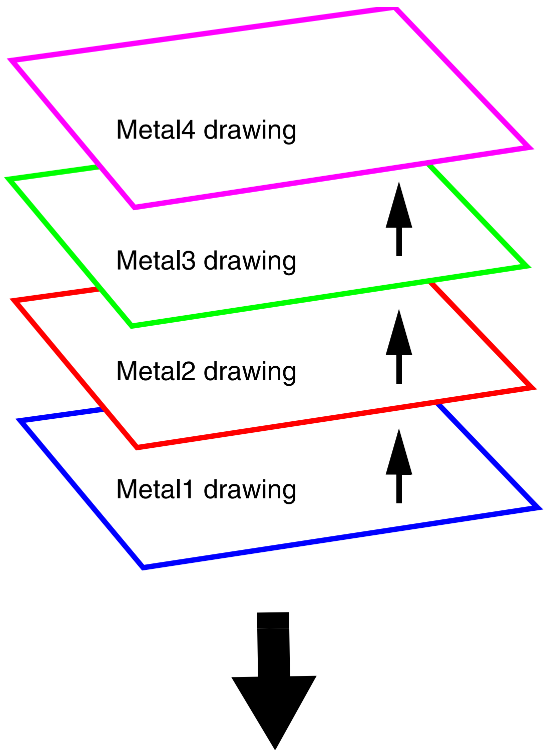
Example

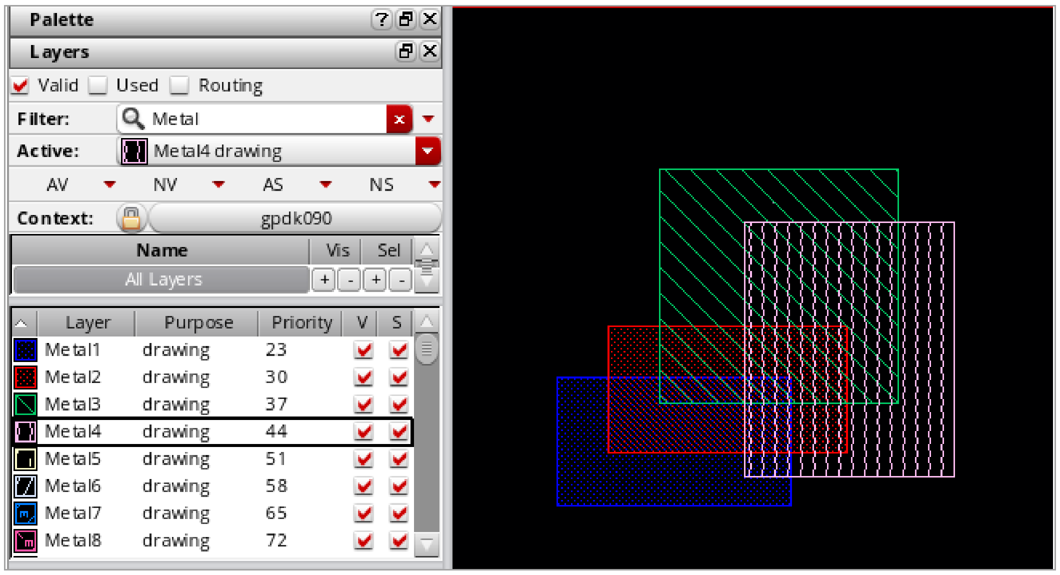
```
techLayerPurposePriorities(
 ; ( layerName     purposeName )
   ( Metal1        drawing     )
   ( Metal2        drawing     )
   ( Metal3        drawing     )
   ( Metal4        drawing     )
) ;techLayerPurposePriorities
```

Defines layer-purpose pair priority based on the order in which the layer-purpose pairs are listed. For example, the layer-purpose pair listed first, `Metal1 drawing`, has the lowest priority, and the layer-purpose pair listed last, `Metal4 drawing`, has the highest priority. The priority assigned to the layer-purpose pairs determines how shapes are rendered in the layout. For example, shapes on the layer-purpose pair with the highest priority are drawn on top of the shapes on all other layer-purpose pairs.

The figure below illustrates how layer-purpose pairs are rendered based on the priority assigned to them. It also includes an example layout that shows shapes drawn on layers

`Metal1` through `Metal4`, with the `Metal1` shape at the bottom and the `Metal4` shape at the top.





✓ Valid ☐ Used ☐ Routing

Filter: Metal

Active: Metal4 drawing

Context: gpdk090

Layer	Purpose	Priority	V	S
Metal1	drawing	23	✓	✓
Metal2	drawing	30	✓	✓
Metal3	drawing	37	✓	✓
Metal4	drawing	44	✓	✓
Metal5	drawing	51	✓	✓
Metal6	drawing	58	✓	✓
Metal7	drawing	65	✓	✓
Metal8	drawing	72	✓	✓

techDisplays

```
techDisplays(
     ( t_layerName
     t_purposeName
     t_packet
     g_visible
     g_selectable
     g_contToChgLay
     g_dragEnable
     g_valid
     ['allowSetToValid t/nil]
     ['description t_description]
     )
     …
     )
```

Description

Defines the display attributes of layer-purpose pairs.

Arguments

t_layerName	Name of the layer.
t_purposeName	Name of the purpose.
t_packet	Name of a packet defined in the display resource file. For more information about display packets and the display resource file, see Chapter 11, “Display Resource File”.
g_visible	Indicates whether the layer is visible in the display device.
g_selectable	Indicates whether the objects drawn on the layer are selectable.
g_contToChgLay	Indicates whether the layer contributes to a changed layer.
g_dragEnable	Indicates whether you can drag a shape created on the layer in the layout editor.
g_valid	Indicates whether the layer appears in the Palette Assistant window.

'allowSetToValid t/nil

The default value of this Boolean attribute is nil. The dumper dumps this attribute only when it is set to t.

'description *t_description*

This string can be used to specify a brief description of the display attributes.

Example

```
techDisplays(
 ; ( layer   Purpose   Packet      Vis   Sel   chgLay   drag   valid
 ;   attribute)
   ( poly1   boundary  redHash_S   t     t     t        t      t
     'allowSetToValid nil 'description "poly boundary layer")
)
```

Sets the display packet to redHash_s for poly1 boundary layer-purpose pair (LPP). In addition, the visible, selectable, change layer, drag enable, and valid attributes are all set to t. The 'allowSetToValid attribute is set to nil.

techLayerProperties

```
techLayerProperties(
     ( t_propName t_layer1 [t_layer2] g_propValue )
     …
) ;techLayerProperties
```

Description

Stores properties for specific user-defined layers. Must be contained within the `layerDefinitions` enclosure.

The following are the properties that can be specified:

single-layer area capacitance	`areaCapacitance`	The capacitance for each square unit in picofarads per square micron; floating-point; used to model wire-to-ground capacitance
two-layer area capacitance	`areaCapacitance`	The capacitance for each square unit in picofarads per square micron; floating-point; used to model capacitance between two facing layers
single-layer edge capacitance	`edgeCapacitance`	The peripheral capacitance in picofarads per micron; floating-point
two-layer edge capacitance	`edgeCapacitance`	The lateral capacitance between two layers in picofarads per micron; floating-point
layer sheet resistance	`sheetResistance`	The resistance for a square of wire in ohms per square
cut layer resistance per cut	`resistancePerCut`	The resistance for a via in ohms per number of cuts
layer height	`height`	The distance from the top of the ground plane to the bottom of the interconnect in user units
layer thickness	`thickness`	The thickness of the layer in user units
shrinkage factor	`shrinkage`	The value to account for shrinkage of interconnect wiring as a result of the etching process in user units

capacitance multiplier	`capMultiplier`	The multiplier for interconnect capacitance to account for increases in capacitance caused by nearby wires. If not specified, applications assume a value of `1`.

Arguments

`t_propName`	The name of the property you want to set. Valid values: `areaCapacitance`, `edgeCapacitance`, `sheetResistance`, `resistancePerCut`, `height`, `thickness`, `shrinkage`, `capMultiplier`
`t_layer1`	The first layer or layer-purpose pair to which the property is applied.
`t_layer2`	The second layer to which the property is applied.
`g_propValue`	The value assigned to the property.

Example

```
techLayerProperties(
    ( sheetResistance        METAL1                        0.02 )
    ( areaCapacitance        (METAL1 drawing)              4.1e-4 )
    ( areaCapacitance        METAL1        METAL2          2.36-5 )
    ( edgeCapacitance        METAL1                        3.0e-5 )
    ( edgeCapacitance        METAL1        METAL2          5.0e-6 )
) ;techLayerProperties
```

techDerivedLayers

With `techDerivedLayers`, you can specify:

- Purpose-Aware–Derived Layers
- Sizing-Derived Layers
- Area-Restricted Derived Layers
- Color-Derived Layers
- Two-Layer–Derived Layers

Purpose-Aware–Derived Layers

```
techDerivedLayers(
     ( tx_derivedLayer x_derivedLayerNum ( tx_layer 'select tx_purpose ) )
     …
) ;techDerivedLayers
```

Description

Defines layers derived by pairing a layer with a purpose. Purpose-aware derived layers allow selection of a layer with the specified purpose for purpose-aware constraint application.

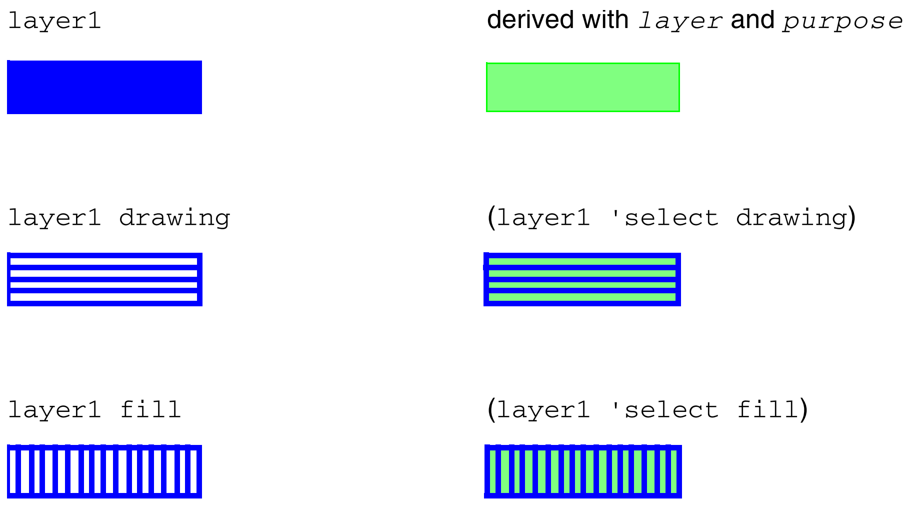
Arguments

tx_derivedLayer	Derived layer name. Valid values: Any string; layer names must be unique
x_derivedLayerNum	Derived layer number. Valid values: Any number not reserved or already assigned
tx_layer	The layer to be paired with a purpose. Valid values: Layer name or layer number
`'select`	The operator indicating that the layer is to be paired with a purpose.
tx_purpose	The purpose to be paired with the layer. Valid values: Any valid purpose that is not reserved

Example

```
techDerivedLayers(
   ("derivedLayerPurpose1" 1000 ("metal1" 'select "drawing"))
   ("derivedLayerPurpose2" 1001 ("metal1" 'select "fill"))
)
```

Defines a derived layer named `derivedLayerPurpose1` with layer number `1000`, created by selecting layer `metal1` and purpose `drawing`. It also defines a derived layer named `derivedLayerPurpose2` with layer number `1001`, created by selecting layer `metal1` and purpose `fill`.



Sizing-Derived Layers

```
techDerivedLayers(
     ( tx_derivedLayer x_derivedLayerNum( tx_layer s_sizeOp g_value ) )
     …
) ;techDerivedLayers
```

Description

Defines layers derived from sizing other layers.

Arguments

tx_derivedLayer	The derived layer. Valid values: The derived layer name or number
x_derivedLayerNum	The derived layer number. Valid values: Any number not reserved or already assigned
tx_layer	The layer to be sized. Valid values: The layer name or layer number
s_sizeOp	The sizing operation to perform on the layer. Valid values: `'growHorizontal`, `'growVertical`, `'shrinkHorizontal`, `'shrinkVertical`, `'grow`, `'shrink`
g_value	The amount by which to size the layer. Valid values: The distance by which the layer grows or shrinks, in user units.

Example

```
techDerivedLayers(
     (metal1Sized  20001  ( "Metal1" 'grow  0.01))
)
```

Defines a derived layer named `metal1sized` with layer number `20001`, with all shapes on `Metal1` grown by 0.01um.

Area-Restricted Derived Layers

```
techDerivedLayers(
     ( tx_derivedLayer ( tx_layer 'area x_area ) )
     …
) ;techDerivedLayers
```

Description

Defines derived layers with restricted area.

Arguments

tx_derivedLayer	The derived layer. Valid values: The derived layer name or number
tx_layer	The layer whose area is to be restricted. Valid values: The layer name or layer number
'area	Specifies that the area is to be restricted.
x_area	The area restriction, specified as follows: > | < *n* where, >*n* specifies that the area must be larger than *n*, the number specified, and <*n* specifies that the area must be smaller than *n*, the number specified.

Example

```
techDerivedLayers(
   ("derivedLayer1" 1000 ("metal1" 'area <5))
)
```

Defines a derived layer named `derivedLayer1` with layer number `1000` by restricting the area to less than 5.

Color-Derived Layers

```
techDerivedLayers(
      ( t_derivedLayerName x_derivedLayerNum
           ( tx_layerName 'color t_maskColor )
           ['locked | 'unlocked]
      )
      …

) ;techDerivedLayers
```

Description

(ICADVM20.1 Only – 95511) Creates a derived layer and copies to it shapes based on their color mask and color locked state, if specified.

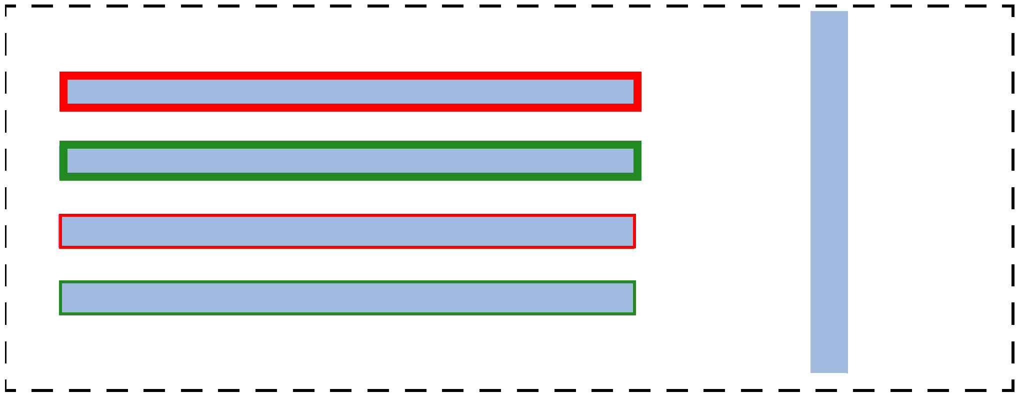
Arguments

t_derivedLayerName

	The derived layer name. Valid values: The derived layer name
`x_derivedLayerNum`	The derived layer number. Valid values: Any number not reserved or already assigned
`tx_layer`	The layer to be paired with a purpose. Valid values: Layer name or layer number
`'color` *`t_maskColor`*	The color of the shape that needs to be copied to the derived layer: `any`, `mask1Color`, `mask2Color`, `mask3Color`, and `grayColor` (uncolored shape).
`'locked` | `'unlocked`	The lock status. By default, all shapes are copied to the derived layer, irrespective of whether they are locked or unlocked. ■ `'locked`: Only locked shapes are copied to the derived layer. ■ `'unlocked`: Only unlocked shapes are copied to the derived layer.

Example

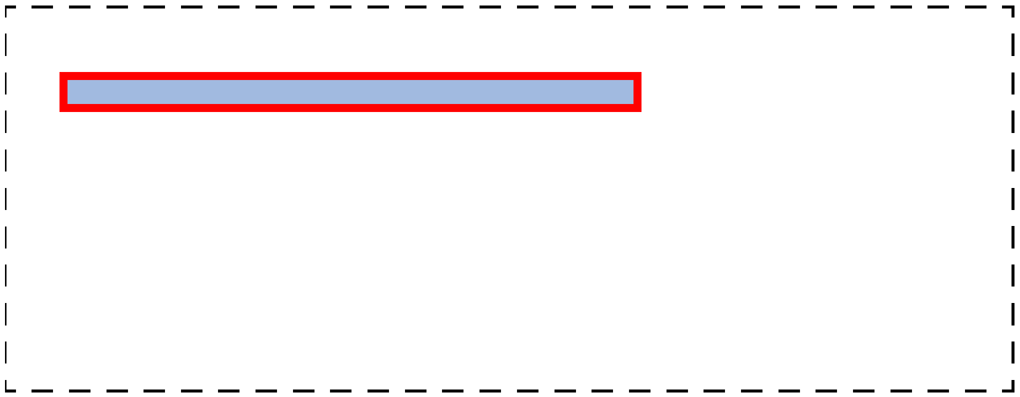
The `metal1` layer has both locked and unlocked colored shapes and one gray shape, as shown below. The various scenarios explain how shapes can be copied to the derived layer.



Scenario 1

```
techDerivedLayers(
    ( metal1_RedLocked 131 ( metal1 'color  mask1Color ) 'locked )
)
```

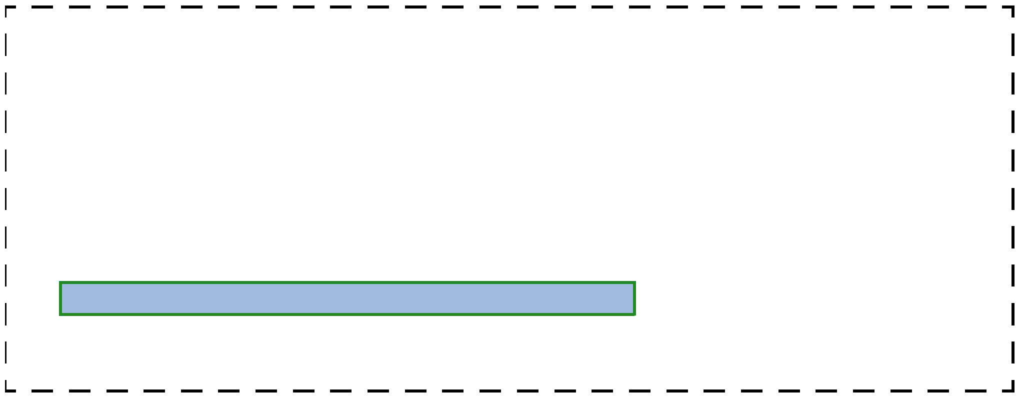
Creates a derived layer named `metal1_RedLocked` with layer number `131` and copies to it the locked `mask1Color` shape, as shown below.



Scenario 2

```
techDerivedLayers(
    ( metal1_GreenUnlocked 131 ( metal1 'color  mask2Color ) 'unlocked )
)
```

Creates a derived layer named `metal1_GreenUnlocked` with layer number `131` and copies to it the unlocked `mask2Color` shape, as shown below.



Scenario 3

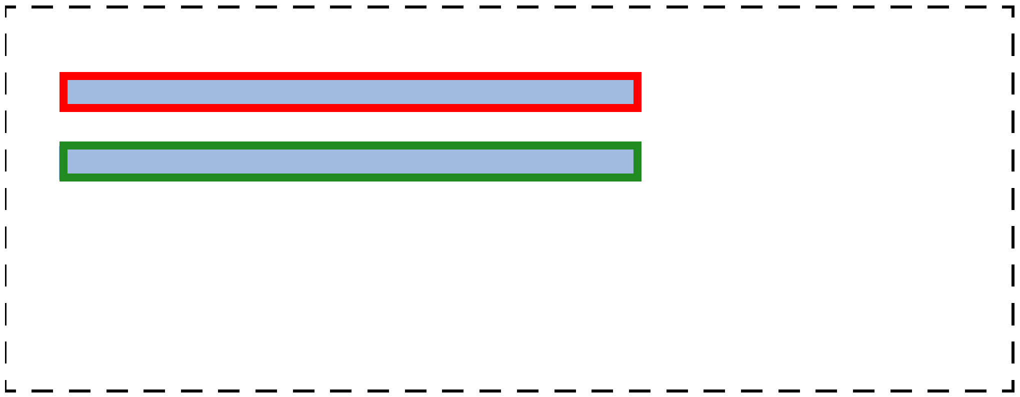
```
techDerivedLayers(
    ( metal1_Gray 131 ( metal1 'color grayColor ) 'unlocked )
)
```

Creates a derived layer named `metal1_Gray` with layer number `131` and copies to it the unlocked `grayColor` shape, as shown below.

Scenario 4

```
techDerivedLayers(
    ( metal1_Locked 131 ( metal1 'color  any ) 'locked )
)
```

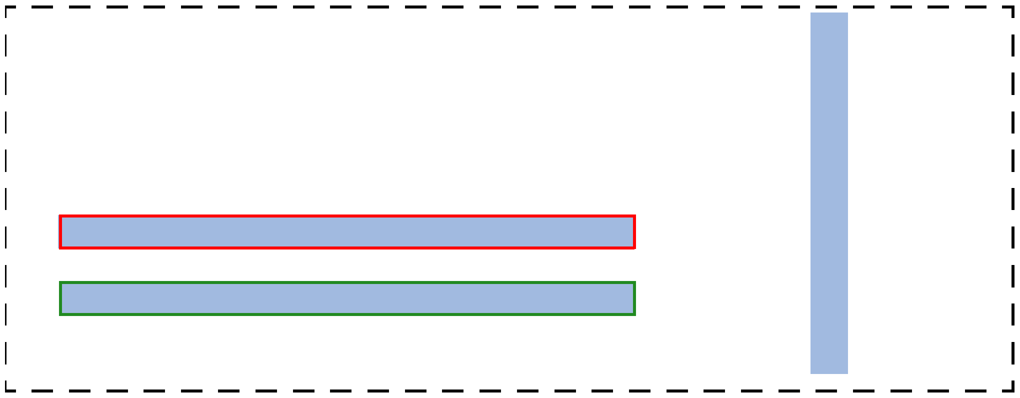
Creates a derived layer named `metal1_Locked` with layer number `131` and copies to it all locked shapes, as shown below.



Scenario 5

```
techDerivedLayers(
    ( metal1_UnLocked 131 ( metal1 'color any ) 'unlocked )
)
```

Creates a derived layer named `metal1_UnLocked` with layer number `131`and copies to it all unlocked shapes, as shown below.



Two-Layer–Derived Layers

```
techDerivedLayers(
     ( t_derivedLayerName x_derivedLayerNum
          ( tx_layer1 s_op tx_layer2 )
          [ x_count | t_rangeVal ]
          [ 'diffNet | 'sameNet ]
          [ 'exclusive ]
     )
     …
) ;techDerivedLayers
```

Description

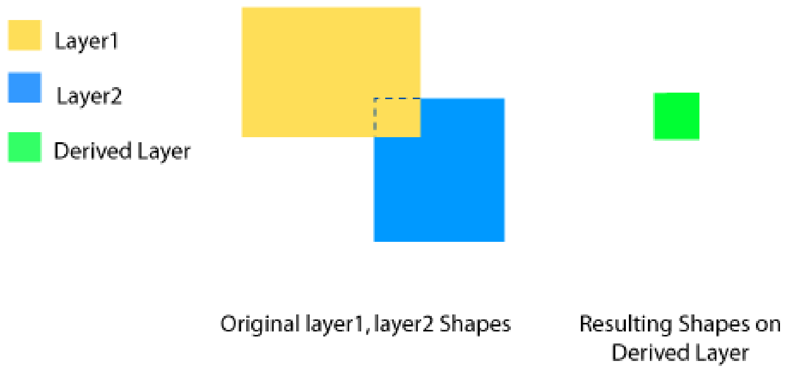
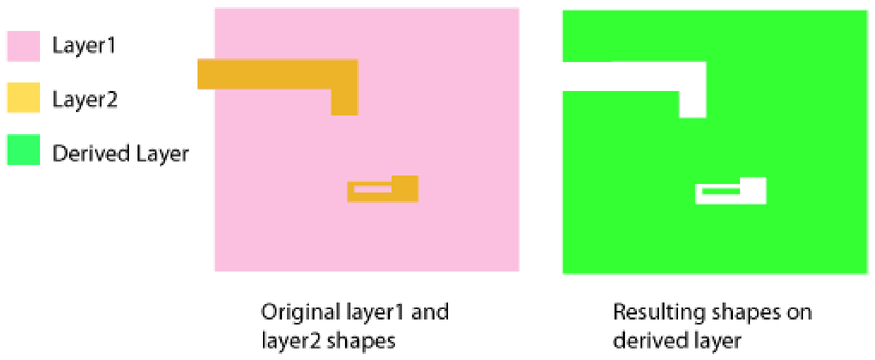
Defines derived layers to be used throughout the technology file. A derived layer is the result of performing a logical operation on two layers.

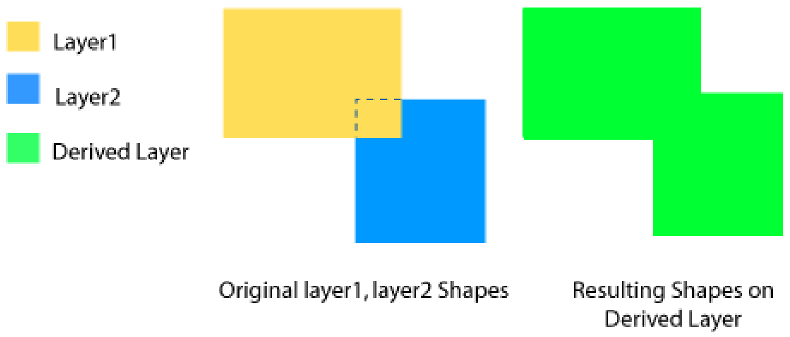
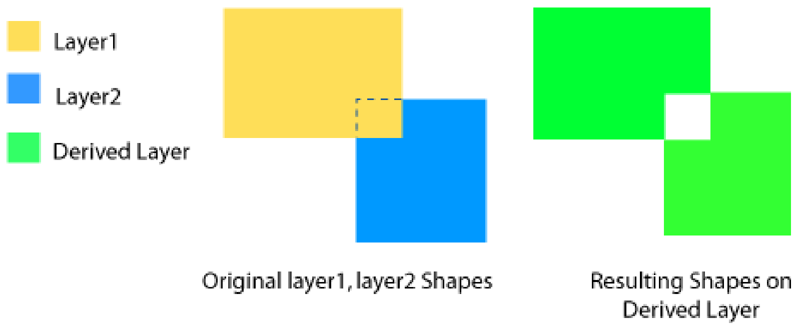
Arguments

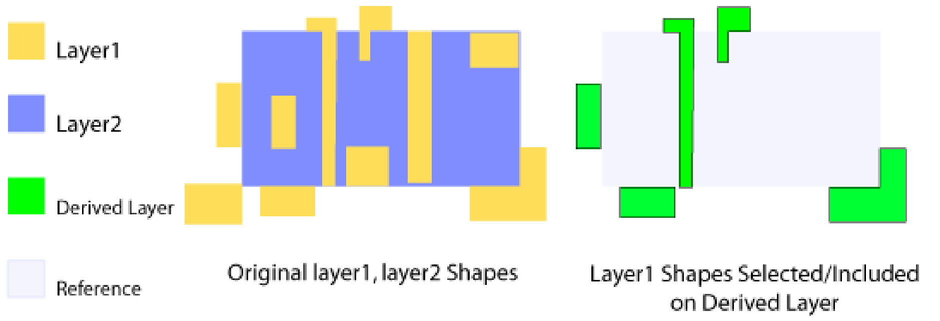
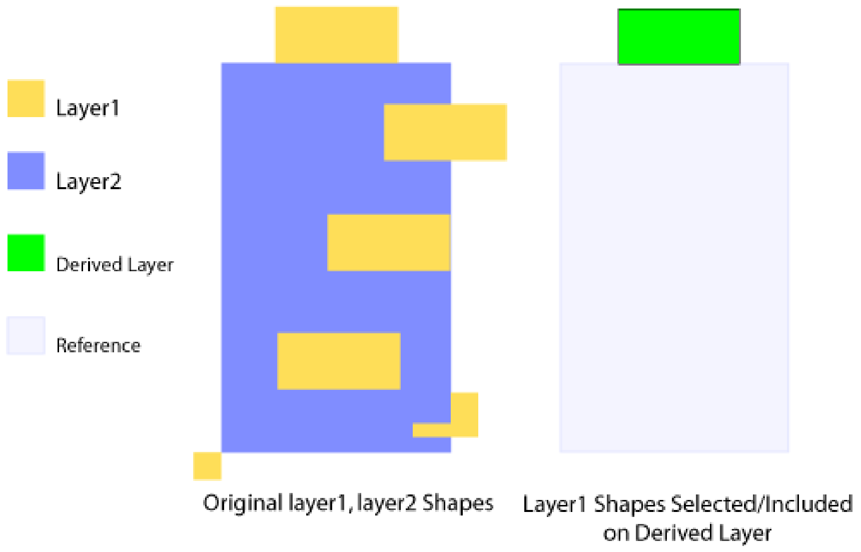
t_derivedLayerName	The name to apply to the derived layer. Valid values: The derived layer name or number
x_derivedLayerNum	The layer number for the derived layer. Valid values: The layer name or layer number
tx_layer1	The first layer used to create the derived layer. Valid values: The layer name or layer number
s_op	The logical operation to perform on the two layers to create the derived layer. Only one logical operator can be used in the creation of a derived layer. Valid values: `'and`, `'or`, `'not`, `'xor`, `'butting`, `'buttOnly`, `'coincident`, `'coincidentOnly`, `'buttingOrCoincident`, `'overlapping`, `'buttingOrOverlapping`, `'touching`, `'inside`, `'outside`, `'avoiding`, `'straddling`, `'enclosing` `'and`, `'or`, `'not`, and `'xor` represent standard logical operations. The other operators are explained in Operator Definitions on page 74.
tx_layer2	The second layer used to create the derived layer. Valid values: The layer name or layer number

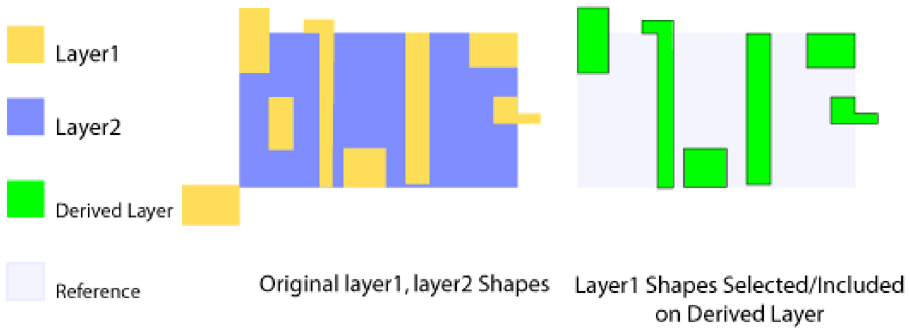
`x_count`	The number of times contact must be made between the shapes on the two layers forming the derived layer. Valid values: Any non-negative integer
`t_rangeVal`	A range of the number of times contact can be made between the shapes on the two layers forming the derived layer. Valid values: `"< `*`n`*`"`, `"≤ `*`n`*`"`, `"> `*`n`*`"`, `"≥1 `*`n`*`"`, `"[`*`n1 n2`*`]"`, `"(`*`n1 n2`*`)"`, `"[`*`n1 n2`*`)"`, `"(`*`n1 n2`*`]"` where, *`n`* is the count, *`n1`* is the lower number in a range, *`n2`* is the upper number in a range, `[]` indicates that the range is inclusive of *`n1`* and *`n2`*, and `()` indicates that the range is exclusive of *`n1`* and *`n2`*. For example, ■ `"[1 5]"` includes values 1, 2, 3, 4, 5 ■ `"(1 5)"` includes values 2, 3, 4 ■ `"[1 5)"` includes values 1, 2, 3, 4 ■ `"(1 5]"` includes values 2, 3, 4, 5
`'diffNet`	The shapes on *`layer1`* and *`layer2`* must be on different nets. If not specified, connectivity is ignored.
`'sameNet`	The shapes on *`layer1`* and *`layer2`* must be on the same net. If not specified, connectivity is ignored.
`'exclusive`	This specification must define the only derived layer relationship between shapes on *`layer1`* and shapes on *`layer2`*.

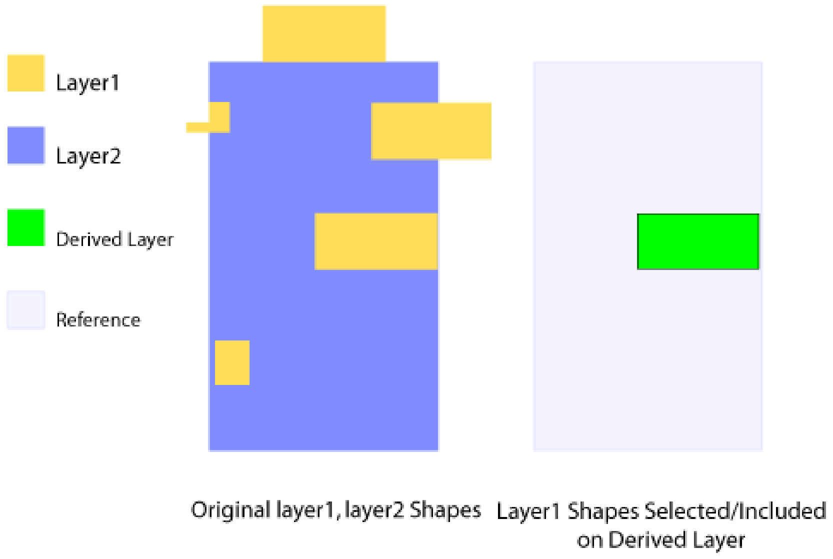
Operator Definitions

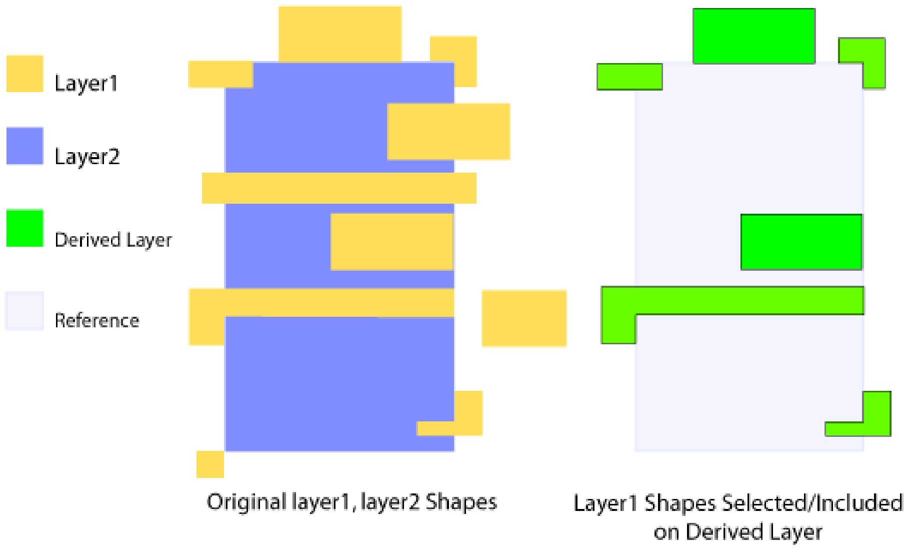
Operator	Definition
`'and`	The `'and` layer operation requires two layers to be specified. This operation creates, on the derived layer, shapes that correspond to the intersecting areas on layer1 and layer2. The following figure is an example of a shape that is included in a derived layer when this type of layer operation is used.  Layer1 Layer2 Derived Layer Original layer1, layer2 Shapes Resulting Shapes on Derived Layer
`'not`	The `'not` layer operation requires two layers to be specified. This operation creates, on the derived layer, shapes that correspond to the inversion of layer2 shapes and layer1. The following figure is an example of a shape included in a derived layer when this type of layer operation is used.  Layer1 Layer2 Derived Layer Original layer1 and layer2 shapes Resulting shapes on derived layer

Operator	Definition
`'or`	The `'or` layer operation requires two layers to be specified. This operation creates, on the derived layer, shapes that correspond to a union of the shapes on layer1 and layer2. The following figure is an example of a shape included in a derived layer when this type of layer operation is used.  Layer1 Layer2 Derived Layer Original layer1, layer2 Shapes Resulting Shapes on Derived Layer
`'xor`	The `'xor` layer operation requires two layers to be specified. This operation selects the non-overlapping areas of shapes on layer1 and layer2, that is, it excludes the overlapping portions of the shapes. The following figure is an example of shapes included in a derived layer when this type of layer operation is used.  Layer1 Layer2 Derived Layer Original layer1, layer2 Shapes Resulting Shapes on Derived Layer

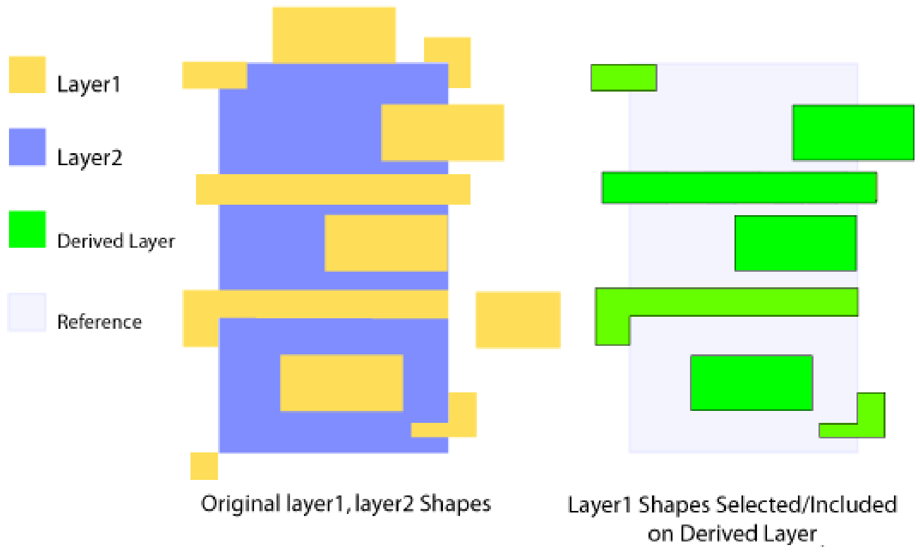
Operator	Definition
`'butting`	The `'butting` layer operation requires two layers to be specified. This operation selects shapes on layer1 that abut with shapes on layer2, that is, it selects shapes with coincident edges and some common area (but not all area in common). The following figure is an example of shapes included in a derived layer when this type of layer operation is used.  Layer1 Layer2 Derived Layer Reference Original layer1, layer2 Shapes Layer1 Shapes Selected/Included on Derived Layer
`'buttOnly`	The `'buttOnly` layer operation requires two layers to be specified. This operation selects shapes on layer1 that abut, but do not overlap shapes on layer2. The following figure is an example of a shape included in a derived layer when this type of layer operation is used.  Layer1 Layer2 Derived Layer Reference Original layer1, layer2 Shapes Layer1 Shapes Selected/Included on Derived Layer

Operator	Definition
`'coincident`	The `'coincident` layer operation requires two layers to be specified. This operation selects shapes on layer1 that have coincident edges with shapes on layer2. A shape on layer1 is coincident if any of its edges is coincident with an edge of a shape on layer2 and the areas of the two shapes at that edge overlap. Other parts of the shape on layer1 can also overlap the shape on layer2. The following figure is an example of shapes included in a derived layer when this type of layer operation is used.  Layer1 Layer2 Derived Layer Reference Original layer1, layer2 Shapes Layer1 Shapes Selected/Included on Derived Layer

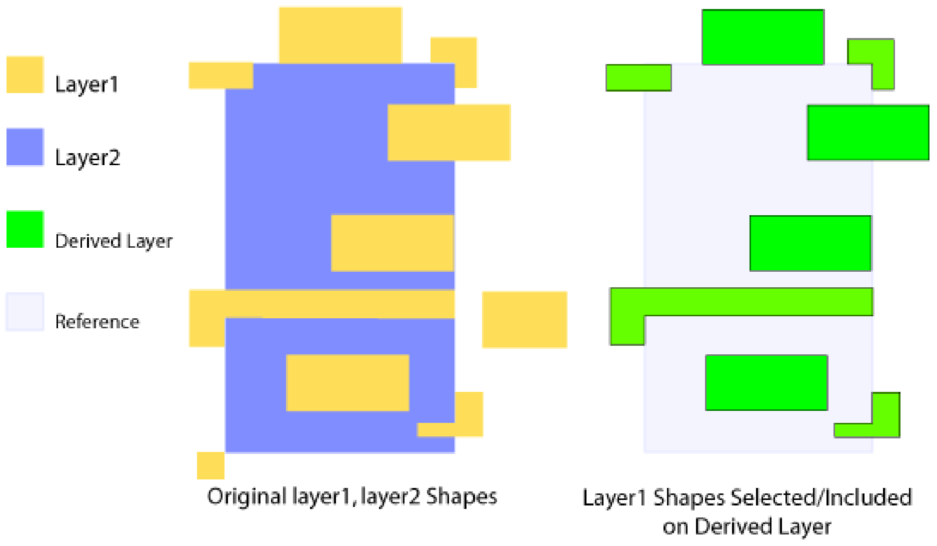
Operator	Definition
`'coincidentOnly`	The `'coincidentOnly` layer operation requires two layers to be specified. This operation selects shapes on layer1 that have at least one coincident edge with a shape on layer2 and the area of the layer2 shape is completely covered by the layer1 shape. This is different from the `'coincident` layer operation in that no portion of the layer1 shape can lie outside of the layer2 shape. The following figure is an example of a shape included in a derived layer when this type of layer operation is used.  Layer1 Layer2 Derived Layer Reference Original layer1, layer2 Shapes Layer1 Shapes Selected/Included on Derived Layer

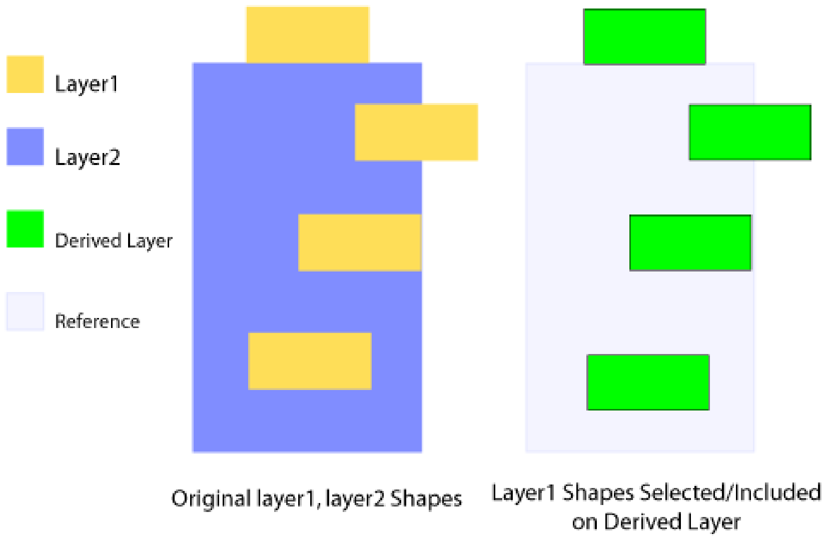
Operator	Definition
`'buttingOrCoincident`	
	The `'buttingOrCoincident` layer operation requires two layers to be specified. This operation selects shapes on layer1 that either abut or are coincident with shapes on layer2. Abutting and coincidence are defined as any edge-to-edge coincidence, regardless of whether the shapes overlap at the point of coincidence. The following figure is an example of shapes included in a derived layer when this type of layer operation is used. For more information, see `'butting` and `'coincident`. 

Layer1
Layer2
Derived Layer
Reference
Original layer1, layer2 Shapes
Layer1 Shapes Selected/Included on Derived Layer

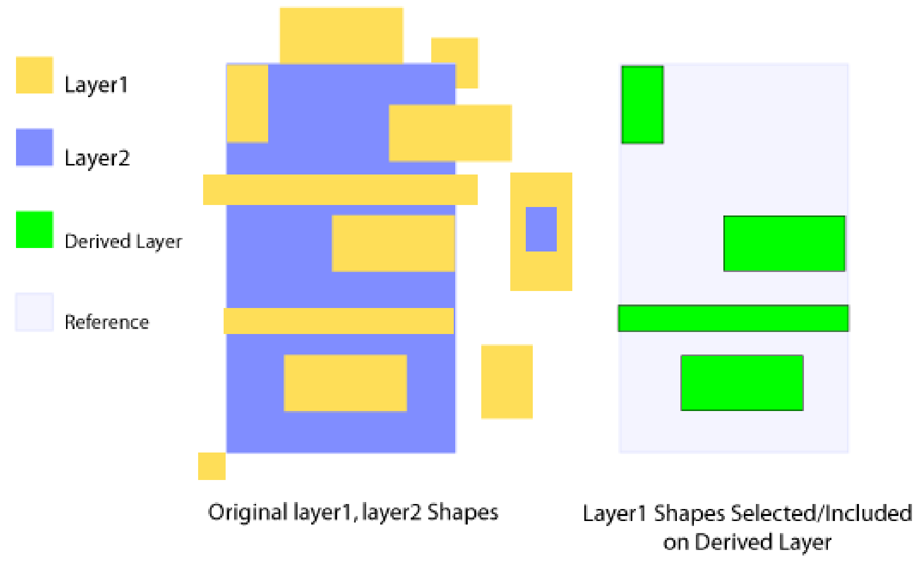
Operator	Definition
`'overlapping`	The `'overlapping` layer operation requires two layers to be specified. This operation selects shapes on layer1 that overlap shapes on layer2. Overlap is defined as any area common to shapes on both layers. A layer1 shape overlaps if it has any area in common with a shape on layer2. The following figure is an example of shapes included in a derived layer when this type of layer operation is used.  Layer1 Layer2 Derived Layer Reference Original layer1, layer2 Shapes Layer1 Shapes Selected/Included on Derived Layer

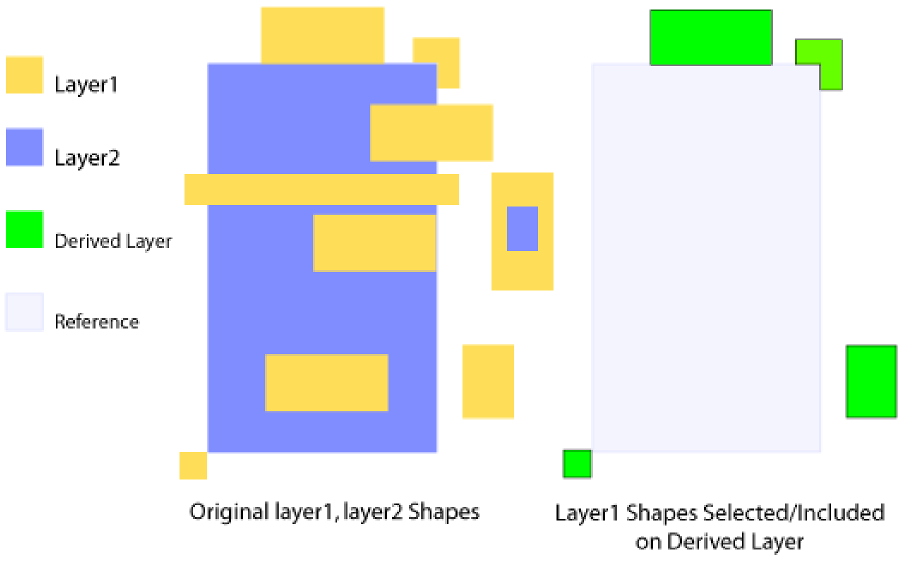
Operator	**Definition**
`'buttingOrOvelapping`	The `'buttingOrOverlapping` layer operation requires two layers to be specified. This operation selects shapes on layer1 that abut or overlap shapes on layer2. Overlap is defined as a common area between shapes. The following figure is an example of shapes included in a derived layer when this type of layer operation is used. For more information, see `'butting` and `'overlapping`. **Note:** The `'buttingOrOverlapping` layer operation is the same as the `'touching` layer operation, but the latter does not support optional parameters.

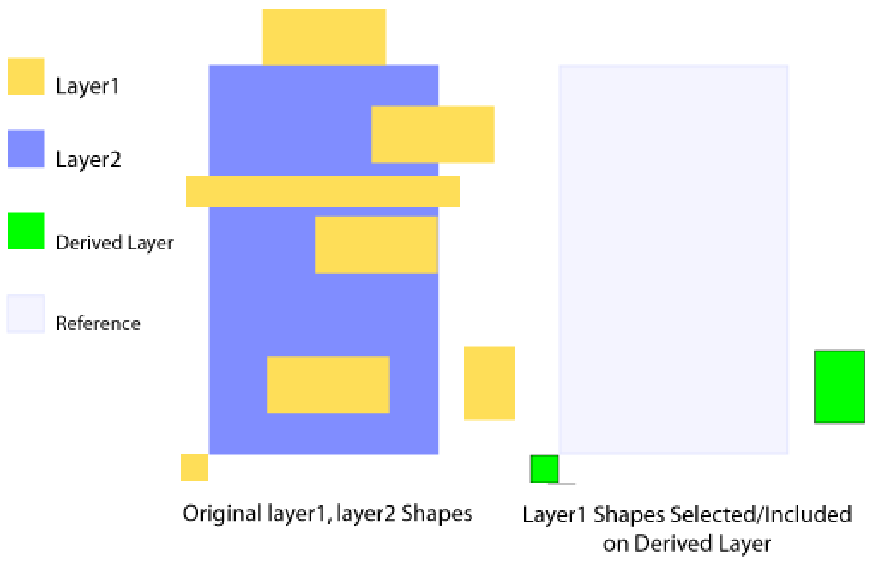

Layer1
Layer2
Derived Layer
Reference
Original layer1, layer2 Shapes
Layer1 Shapes Selected/Included on Derived Layer

Operator	Definition
`'touching`	The `'touching` layer operation requires two layers to be specified. This operation selects shapes on layer1 that are completely inside, partially inside, or abutting a shape on layer2. The following figure is an example of shapes included in a derived layer when this type of layer operation is used. **Note:** None of the optional parameters can be specified when a derived layer definition contains the `'touching` operator. To specify any of these optional parameters, use `'buttingOrOvelapping`. 

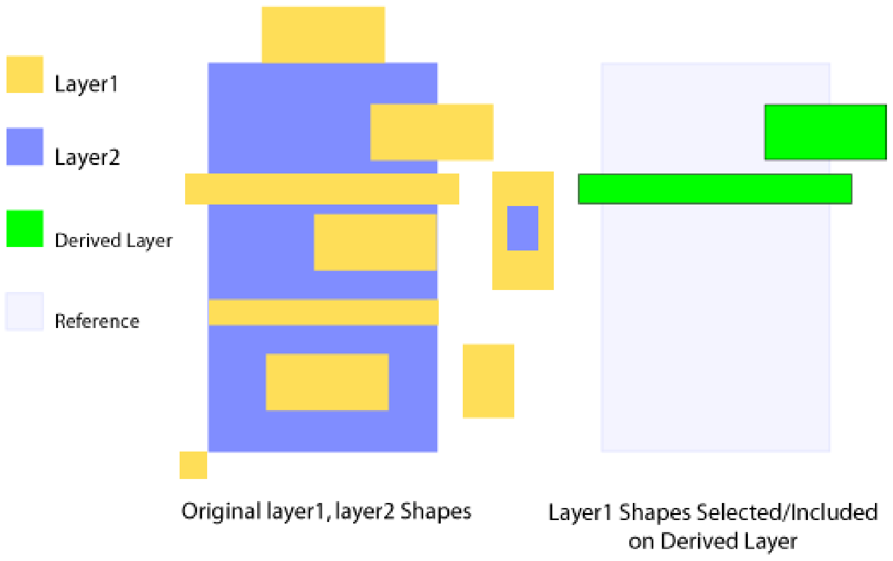
Layer1
Layer2
Derived Layer
Reference
Original layer1, layer2 Shapes
Layer1 Shapes Selected/Included on Derived Layer

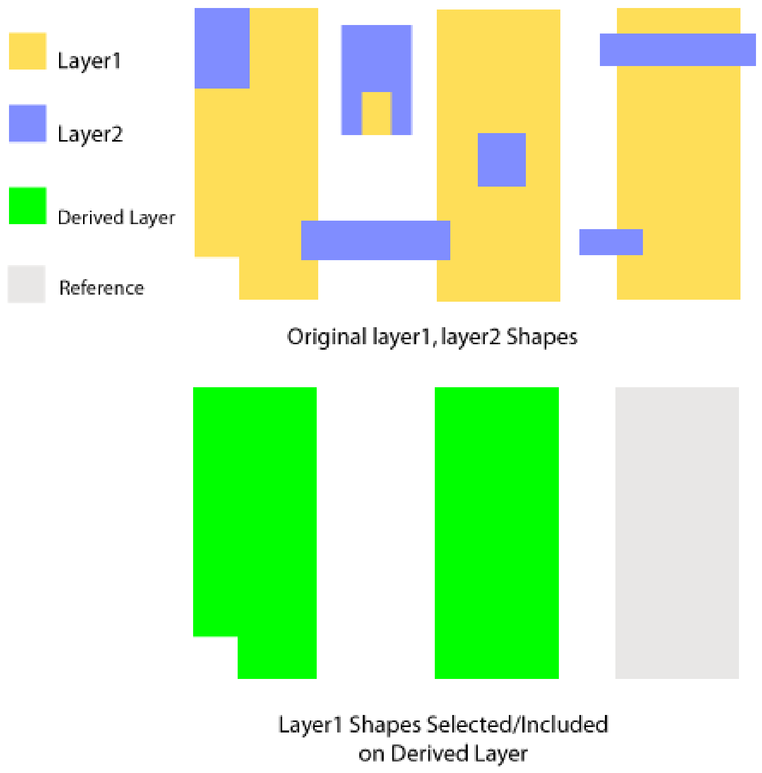
Operator	Definition
`'inside`	The `'inside` layer operation requires two layers to be specified. This operation selects shapes on layer1 that are completely inside a shape on layer2. Coincident edges are allowed, but layer1 shapes that are not completely inside layer2 shapes are not selected. The difference between this and the `'touching` layer operation is that this operation does not include the abutting shapes that are outside the layer2 shapes and overlapping shapes that are only partially inside the layer2 shapes. The following figure is an example of shapes included in a derived layer when this type of layer operation is used.  Layer1 Layer2 Derived Layer Reference Original layer1, layer2 Shapes Layer1 Shapes Selected/Included on Derived Layer

Operator	Definition
`'outside`	The `'outside` layer operation requires two layers to be specified. This operation selects shapes on layer1 that are completely outside a shape on the layer2. Coincident edges are allowed. The following figure is an example of shapes included in a derived layer when this type of layer operation is used.  Layer1 Layer2 Derived Layer Reference Original layer1, layer2 Shapes Layer1 Shapes Selected/Included on Derived Layer

Operator	Definition
`'avoiding`	The `'avoiding` layer operation requires two layers to be specified. This operation selects shapes on layer1 that are completely outside of, and have no abutting edges with, shapes on layer2. The following figure is an example of shapes included in a derived layer when this type of layer operation is used. 

Layer1
Layer2
Derived Layer
Reference
Original layer1, layer2 Shapes
Layer1 Shapes Selected/Included on Derived Layer

Operator	Definition
`'straddling`	The `'straddling` layer operation requires two layers to be specified. This operation selects shapes on layer1 that are only partially inside shapes on layer2. The following figure is an example of shapes included in a derived layer when this type of layer operation is used.  Layer1 Layer2 Derived Layer Reference Original layer1, layer2 Shapes Layer1 Shapes Selected/Included on Derived Layer

Operator	Definition
`'enclosing`	The `'enclosing` layer operation requires two layers to be specified. This operation selects shapes on layer1 that completely cover at least one shape on layer2. In addition, a count that governs how many layer2 shapes must be enclosed by a layer1 shape for the layer1 shape to be selected can be specified. The following figure is an example of shapes included in a derived layer when this type of layer operation is used. 

Layer1
Layer2
Derived Layer
Reference
Original layer1, layer2 Shapes
Layer1 Shapes Selected/Included on Derived Layer

LPP Validity

A layer-purpose pair (LPP) is created from a combination of a single layer and a single purpose object defined in the technology database.

There are several ways of controlling the validity of an LPP. The `valid` attribute of the `LPId` can be set directly to `true` or `false`. However, the effective validity of the LPP depends on the `valid` attribute set for the parent layer, the parent purpose, and the LPP itself. Therefore, the effective LPP validity is equal to layer `valid` && purpose `valid` && lp base `valid`.

The layer `valid` attribute contributes to the validity of all layer-purpose pairs that the layer is part of. Similarly, the purpose `valid` attribute contributes to the validity of all layer-purpose pairs that the purpose is part of. The default values of layer and purpose `valid` attributes are `true`.

The ability to set the valid attributes of a layer, purpose, or layer-purpose pair can be controlled by the following attributes:

- `allowSetToValid`: This attribute is persistent.
- `allowSetToValidInSession`: This attribute is not persistent and applies only during the current session.

These attributes exist on the layer, purpose, and layer-purpose pair objects.

If an object's `allowSetToValid` or `allowSetToValidInSession` attribute is `false`, the `valid` attribute may not be set to `true`.

The `allowSetToValid` and `allowSetToValidInSession` attributes can always be set to `false`. However, the attributes can only be set to `true` when the parent technology database is opened in the "write" or "append" mode.

LPP Validity Attributes

- `layerId~>valid`

  Gets or sets the layer `valid` attribute. The attribute can be changed to `true` only when the layer attributes `allowSetToValid` and `allowSetToValidInSession` are `true`. The default value is `true`.

- `layerId~>allowSetToValid`

  Gets or sets the layer `allowSetToValid` attribute. The attribute can be changed to `true` only when the technology database of the corresponding layer is opened in "append" or "write" mode. This attribute is persistent and the default value is `true`.

- `layerId~>allowSetToValidInSession`

  Gets or sets the layer `allowSetToValidInSession` attribute. The attribute can be changed to `true` only when the technology database of the corresponding layer is opened in "append" or "write" mode. This attribute is not persistent and the default value is `true`.

- `purposeDefId~>valid`

  Gets or sets the purpose `valid` attribute. The attribute can be changed to `true` only when the purpose attributes `allowsetToValid` and `allowSetToValidInSession` are `true`. The default value is `true`.

- `purposeDefId~>allowSetToValid`

  Gets or sets the purpose `allowSetToValid` attribute. The attribute can be changed to `true` only when the technology database of the corresponding purpose is opened in "append" or "write" mode. This attribute is persistent and the default value is `true`.

- `purposeDefId~>allowSetToValidInSession`

  Gets or sets the purpose `allowSetToValidInSession` attribute. The attribute can be changed to `true` only when the technology database of the corresponding purpose is opened in "append" or "write" mode. This attribute is not persistent and the default value is `true`.

- `lpId~>allowSetToValid`

  Gets or sets the LPP `allowSetToValid` attribute. The attribute can be changed to `true` only when the technology database of the corresponding LPP is opened in "append" or "write" mode. This attribute is persistent and the default value is `true`.

- `lpId~>allowSetToValidInSession`

  Gets or sets the LPP `allowSetToValidInSession` attribute. The attribute can be changed to `true` only when the technology database of the corresponding LPP is opened in "append" or "write" mode. This attribute is not persistent and the default value is `true`.

- `techIsLPValidBase(lpId)`

  Checks if the LPP base `valid` attribute is `true`.

- `lpId~>valid`

  Gets the effective LPP `valid` attribute. Effective LPP `valid` attribute is equal to layer `valid` && purpose `valid` && LPP base `valid`.

Sets the LPP base `valid` attribute. The LPP base `valid` attribute can be changed to `true` only when the following conditions are met:

- ❑ The LPP attributes `allowSetToValid` and `allowSetToValidInSession` are `true`.
- ❑ The `allowSetToValid` and `allowSetToValidInSession` attributes for the corresponding layer are `true`.
- ❑ The `allowSetToValid` and `allowSetToValidInSession` attributes for the corresponding purpose are `true`.

5

Technology File Layer Attributes

The `layerRules` section of the technology file specifies attributes for user-defined layers.

Note: Do not redefine system-reserved layers or purposes. The software discards any such customization.

This chapter contains the following topics:

- stampLabelLayers
- labelLayers
- cutClasses

layerRules

```
layerRules()
```

Description

Contains layer rules. All subsections specifying layer rules must be enclosed within the parentheses of this section.

Important

Layer attributes must be assigned only to user-defined layers. Layer attributes if applied to system-reserved layers are ignored.

The *`layerOrder`* values should be specified in the order of the process stack, and not in the order in which layers are manufactured. For example, even if a cut layer is manufactured after its top metal layer (for example, if the process uses self-aligned vias), the *`layerOrder`* number of the cut layer should be less than that of the metal layer.

Related Topics

layerRules

equivalentLayers

```
layerRules(
     equivalentLayers(
          ( lt_layer … )
          …
     ) ;equivalentLayers

) ;layerRules
```

Description

Specifies layers that are physically and electrically equivalent or that represent the same type of material. It can be used to define layers that connect by overlap rather than through a via.

Arguments

lt_layer	A list of layers. Valid values: The layer name or a list specifying the layer name and purpose

Example

```
equivalentLayers(
    ( "metal1_supply" "metal1" )
)
```

Related Topics

- layerRules

incompatibleLayers

```
layerRules(
     incompatibleLayers(
          ( lt_layer ( lt_layer1 lt_layer2 … ) … )
          …
     ) ;incompatibleLayers
) ;layerRules
```

Description

Lists layers, both physical and derived, incompatible with the specified physical layer. This allows exclusion of layers that do not belong to the technology graph of a technology file. By specifying a list of layer names from other incompatible versions of the process layer stack, a warning is displayed as soon as any of these layers is detected in a technology graph.

Arguments

lt_layer	A physical layer that is incompatible with the layers specified in the list that follows. The layer must be defined in the techLayers subsection. Valid values: The layer name or the layer number
lt_layer1 lt_layer2 …	The derived or physical layers that are incompatible with the specified layer, and, therefore, must be excluded from the specified technology database.

Example

```
incompatibleLayers(
   ( "metal1" ( "metal1_mid" "metal1_top") )
) ;incompatibleLayers
```

Specifies that `metal1` is incompatible with layers `metal1_mid` and `metal1_top`. As a result, these layers are not allowed in a technology graph that contains the technology database in which the `incompatibleLayers` statement is specified. This allows you to model two versions of the process stack by using two technology graphs. Any accidental mixing of these two technology graphs is detected by the software.

Related Topics

- layerRules

functions

```
layerRules(
     functions(
          ( t_layer | x_number t_function [ x_layerOrder ]
               ['trims ( (t_physicalLayer t_derivedLayer) … )]
          )
          …
     ) ;functions
) ;layerRules
```

Description

Defines layer functions (materials) and optionally assigns to layers a number that signifies the layer order. Layers assigned the function `other` or `unknown` and a layer order cannot appear in any other section of the technology file.

Note: `other` and `unknown` are two different ASCII technology filenames for the same material.

Multiple layers can be assigned the same layer order. Layers without an explicitly assigned layer order take the default value `0xffffffffu`. The specified order is used as a proxy for the process stack by some applications such as `lefout`. Therefore, layers in a technology file should be ordered by the given order number, from bottom to top.

Important

The `layerRules` section in the technology file for Virtuoso Space-based Router is slightly more constrained. For example, it defines a unique layer order for each layer function. For more information, see the *Virtuoso Space-based Router User Guide*.

Related Topics

- layerRules

Arguments

t_layer	The layer name. Valid values: The layer name
x_number	The layer number. Valid values: The layer number
t_function	The layer function. Valid values: `buriedN`, `buriedP`, `cut`, `deepNimplant`, `deepNwell`, `deepPimplant`, `deepPwell`, `diff`, `li`, `drill`, `diestack`, `metal`, `mimcap`, `ndiff`, `nimplant`, `nplus`, `nwell`, `other`, `padmetal`, `passivationCut`, `pdiff`, `pimplant`, `pipcap`, `poly`, `pplus`, `pseudoCut`, `pseudoMetal`, `pseudoTrim`, `pwell`, `recognition`, `specialCut`, `stackedMimcap`, `trim`, `trimpoly`, `tsv`, `tsvMetal`, `waveguide`, `wirebond`. **Note:** `pplus` and `pimplant` are two different ASCII technology filenames for the same material. Similarly, `nplus` and `nimplant` are also two different ASCII technology filenames for the same material. *Important* You need the `Virtuoso_Photonics_Option` license to use the `waveguide` functionality. For information about obtaining the required license, contact your local Cadence representative.
x_layerOrder	The order/sequence number for the layer. Valid values: Any integer
t_physicalLayer	The physical layer that is being trimmed by a trim layer.
t_derivedLayer	The derived layer that represents the effective layer after the trim is performed.

- The `mimcap` layer material is used to designate the metal layer of a MIM (Metal/Insulator/ Metal) capacitor that is not directly connected to routers. Therefore, `mimcap` layers should not be added to the router `validLayer` list. A cut layer that connects `mimcap` to the routing metal above it terminates at `mimcap`.
- The `stackedMimcap` layer material is also used to designate the metal layer of a MIM capacitor, and should not be added to the router `validLayer` list. However, several `stackedMimcap` layers can be placed between the routing metal layers above and

below. The same cut layer that connects the routing layer above to the routing layer below can also connect to each `stackedMimcap` type layer between those routing layers.

- The `tsv` (through silicon via) layer material indicates that the cut layer is a through-silicon-via cut layer. It is a vertical electrical connection passing completely through a silicon wafer.
- The `tsvMetal` layer material can connect to a `tsv` layer or a `cut` layer. It is not a routing layer.
- The `padMetal` layer material is the last metal layer on the front side or backside of a chip. It can connect either to a `cut` layer or directly to another metal layer without a `cut` layer in between. It can also connect to a `tsv` layer. It is not a routing layer.
- The `pimplant` and `nimplant` layers are used to generate implant masks. During post data processing, these enable you to filter the `pdiff` and `ndiff` layers, which are usually drawn as a single oxide layer.
- The `passivationCut` layer material represents the cuts in the top insulating layer of the chip, known as the passivation layer. Cuts are made in the passivation layer to allow bond wires to connect to the pads below this layer.
- The `li` (local interconnect) layer material defines a layer that directly contacts another layer or a set of layers without an intermediate layer with a `cut` function.
- The `pdiff` and `ndiff` layer materials are used when implicit oxide definition mask is used. In implicit implantation, no dedicated layer is available for implantation, as shown below:

Layer	Function
M1	“metal”
CO	“cut”
NOD	“ndiff”
POD	“pdiff”
NW	“nwell”
PW	“pwell”

- The `trim` layer material is used to specify a layer that removes material from another layer. For example, a `cutPoly` layer that removes material from a `Poly` layer.

  Note: The trim layer is the preferred method used by Virtuoso Space-based Router to determine the usage of cut poly operations. For technologies that do not support trim layer material, see Specifying Stop Layers in the validLayers Constraint for an alternative method.

- The `pplus` and `nplus` layer materials should be used in combination with an explicit (drawn) oxide definition mask. In explicit implantation, a dedicated layer is available to display the implantation area, as shown below:

Layer	**Function**
M1	“metal”
CO	“cut”
OD	**“diff”**
NP	“nplus”
PP	“pplus”
NW	“nwell”
PW	“pwell”

- The `specialCut` layer material connects one or more layers.

Example 1

```
functions(
    ( NWELL      "nwell"   1  )
    ( PWELL      "pwell"   2  )
    ( NDIFF      "ndiff"   3  )
    ( NPLUS      "nplus"   4  )
    ( PPLUS      "pplus"   5  )
    ( capMetal   "mimcap"  20 )
    ( throughSi  "tsv"     40 )
) ;functions
```

Defines layer functions and assigns a layer order to the layers.

Example 2

```
layerDefinitions(
    techLayers(
        ;( LayerName  Layer#  Abbreviation )
        ;( ---------  ------  ------------ )
         ( Poly         20          PO       )
         ( CutPoly      21          CP       )
    ) ;techLayers
    techDerivedLayers(
        ;( DerivedLayerName       #       composition  )
        ;( ----------------    ------     ------------ )
         ( Gate                  1000      ( Poly  'and  Active  ) )
         ( PolyInterConn         1002      ( Poly  'not  CutPoly ) )
    ) ;techDerivedLayers
) ;layerDefinitions
layerRules(
    functions(
        ;( layer     function [maskNumber])
        ;( -----     -------- ------------)
         ( Poly       "poly"     20 )
         ( CutPoly    "trim"     21  'trims  ( (Poly PolyInterConn) ) )
    ) ;functions
) ;layerRules
```

Defines the `CutPoly` layer by using the `trim` material type. The `CutPoly` layer removes material from the `Poly` layer. The resulting derived layer is called `PolyInterConn`. This is the effective `Poly` layer.

Example 3

```
techDerivedLayers(
    ( m1mask2Locked  1011  ( Metal1         'color  "mask2Color" ) 'locked )
    ( m1mask2Cut     1031  ( m1mask2Locked  'not    TM ) )
) ;techDerivedLayers
functions(
    ( Metal1  "metal"  100 )
    ( TM      "trim"   101  'trims ( (Metal1  m1mask2Cut) ) )
) ;functions
```

Defines an uncolored trim layer, `TM`, that cuts only the locked `maskColor2` shapes on `Metal1`.

Example 4

```
techDerivedLayers(
    ( m1mask1        1011     ( Metal1   'color  mask1Color) )
    ( m1mask2        1012     ( Metal1   'color  mask2Color) )
    ( TMmask1        1021     ( TM       'color  mask1Color) )
    ( TMmask2        1022     ( TM       'color  mask2Color) )
    ( m1mask1Cut     1031     ( m1mask1  'not    TMmask1   ) )
    ( m1mask2Cut     1032     ( m1mask2  'not    TMmask2   ) )
) ;techDerivedLayers
functions(
    ( Metal1  "metal"  100 )
    ( TM      "trim"   101  'trims ( (Metal1 m1mask1Cut) (Metal1 m1mask2Cut) ) )
) ;functions
```

Defines a colored trim layer, `TM`. Trim shapes on `TM` with `mask1Color` cut `mask1Color` shapes on `Metal1` and trim shapes on `TM` with `mask2Color` cut `mask2Color` shapes on `Metal1`.

Related Topics

- layerRules

backsideLayers

```
layerRules(
     backsideLayers(
          tx_layer …
     ) ;backsideLayers

) ;layerRules
```

Description

Specifies the layers that are used on the backside of a wafer. These layers are used to form the bottom layer of through-silicon-vias (TSV).

Arguments

tx_layer	The layers that are used on the backside of a wafer. Valid values: The layer name or the layer number

Example

```
layerRules(
    backsideLayers(
        MB
    ) ;backsideLayers
) ;layerRules
```

Defines layer MB as a backside metal layer.

Related Topics

- layerRules

analysisAttributes

```
layerRules(
     analysisAttributes(
          ( tx_layer
               ['materialName t_materialName]
               ['thickness f_thickness]
               ['conductivity f_conductivity]
               ['permittivity f_permittivity]
               ['lossTangent f_lossTangent]
          )
     ) ;analysisAttributes

) ;layerRules
```

Description

Specifies the names and related attributes of the materials used to manufacture a layer.

Arguments

tx_layer	The layer for which the attributes are specified. Valid values: The layer name or the layer number
t_materialName	The name of the material.
'thickness *f_thickness*	The thickness of the layer in microns.
'conductivity *f_conductivity*	The conductivity of the layer in siemens/meter.
'permittivity *f_permittivity*	The dielectric constant of the layer.
'lossTangent *f_lossTangent*	The loss tangent of the layer.

Example

```
layerRules(
    analysisAttributes(
        ( M1    'materialName   "MX"
                'thickness      23
        )
        ( M2    'materialName   "MX"
                'conductivity   6e7
                'thickness      23
        )
    ) ;analysisAttributes
) ;layerRules
```

Defines material attributes for two layers:

- For `M1` layer: material name `MX` and thickness `23` microns.
- For `M2` layer: material name `MX`, conductivity as `6e7` Siemens/meter, and thickness as `23` microns.

Related Topics

- layerRules

mfgResolutions

```
layerRules(
     mfgResolutions(
          ( tx_layer g_value )
          …
     ) ;mfgResolutions

) ;layerRules
```

Description

Specifies that grid snapping for the specified layer must be a multiple of the specified value.

This constraint (layer manufacturing grid resolution) can be applied to any layer. If it is not specified for a layer, the overall `mfgGridResolution` constraint applies.

Arguments

`tx_layer`	The layer on which the constraint is applied. Valid values: The layer name or the layer number
`g_value`	The manufacturing grid resolution for the layer. **Note:** We recommend that the layer manufacturing resolution be equal to or greater than the manufacturing grid resolution.

Example

```
mfgResolution(("metal1" 0.001000)("poly1" 0.000500))
```

Sets the layer manufacturing grid resolution to `0.001000` for layer `metal1` and to `0.00500` for layer `poly1`.

Related Topics

- layerRules

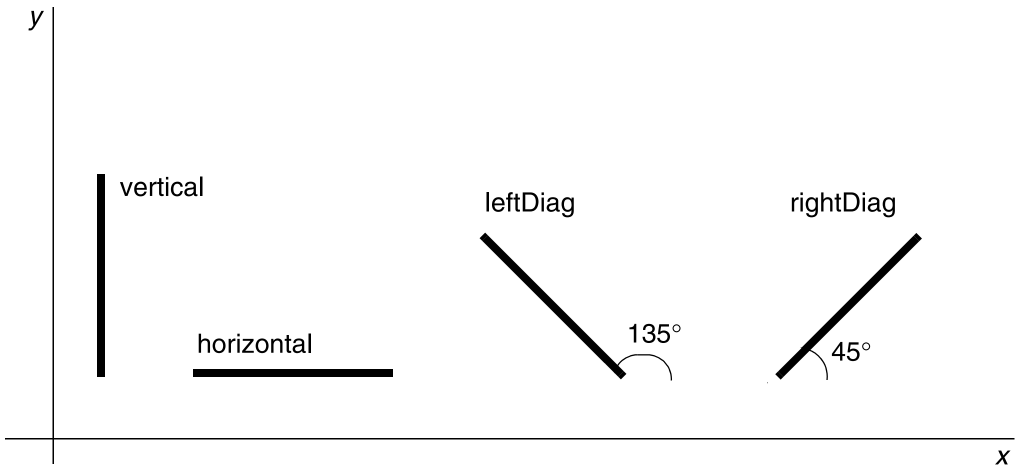
routingDirections

```
layerRules(
     routingDirections(
          ( tx_layer g_direction )
          …
     ) ;routingDirections

) ;layerRules
```

Description

Sets the preferred routing direction for each specified layer.

The direction specified by this layer attribute is the preferred routing direction for the layer. The possible routing directions are illustrated in the figure below:



Arguments

`tx_layer`	The layer on which the attribute is to be applied. Valid values: The layer name or the layer number
`g_direction`	The routing direction for the layer. Valid values: `vertical`, `horizontal`, `leftDiag`, `rightDiag`, `none`, `notApplicable` **Notes:** Specify `none` for routing layers to which you do not want to assign a preferred routing direction or for diffusion and poly layers. Specify `notApplicable` to non-routing layers such as cut layers.

Example

```
layerRules(
    routingDirections(
        ( "metal1" "horizontal")
        ( "metal2" "vertical")
        ( "diff"   "none")
        ( "via1"   "notApplicable")
    )
) ;layerRules
```

Sets the routing direction for `metal1` to `horizontal`, for `metal2` to `vertical`, for `diff` to `none`, and for `via1` to `notApplicable`.

Related Topics

- layerRules

snapPatternDefs

```
layerRules(
     snapPatternDefs(
          (t_name (tx_layer tx_purpose)
               'step g_step
               'stepDirection {"horizontal" | "vertical"}
               ['type {"local" | "global"} ]
               ['offset g_offset]
               ['snappingLayers (
                    ('layer tx_snapLayer …
                    'enclosures f_enclosure …
                    ['purposes tx_snapPurpose …]
                    ['snappingMode {"multiTrackCenter" | "singleTrackCenter"}]
                    ['exceptOverlap l_overlapLPPs]
                    ) … )
               ['trackWidth g_trackWidth]
               ['trackGroups ( ('count n_trackCount
                               'space x_groupSpace
                              ) … )
               ]
          )
     ) ;snapPatternDefs

) ;layerRules
```

Description

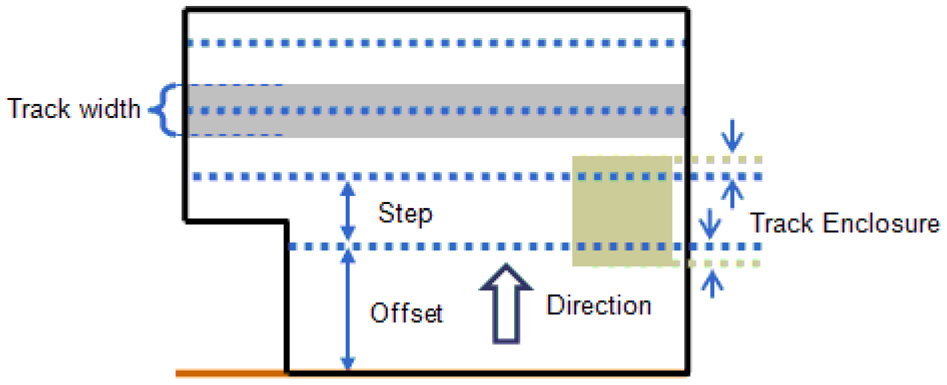
(ICADVM20.1 Only – 95512) Specifies the grid information stored in the technology database for a specific layer-purpose pair (LPP). Each snap pattern definition must have a unique name and be associated with a unique LPP. Snap pattern definitions can be local or global.

Video

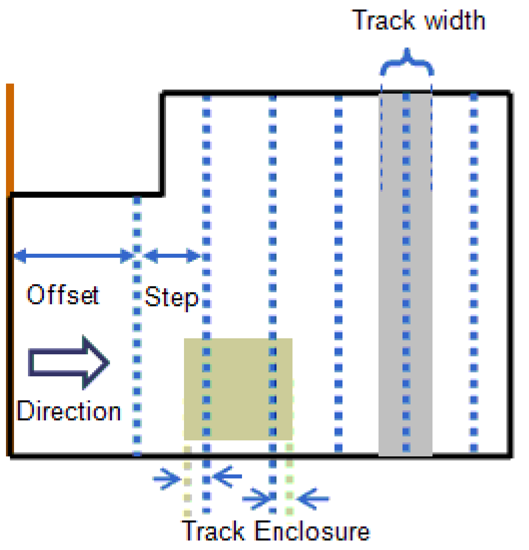
For a quick overview of snap pattern definitions, see Snap Pattern Definition in the TechFile.

A shape drawn on an LPP with a snap pattern definition is referred to as a snap pattern shape.

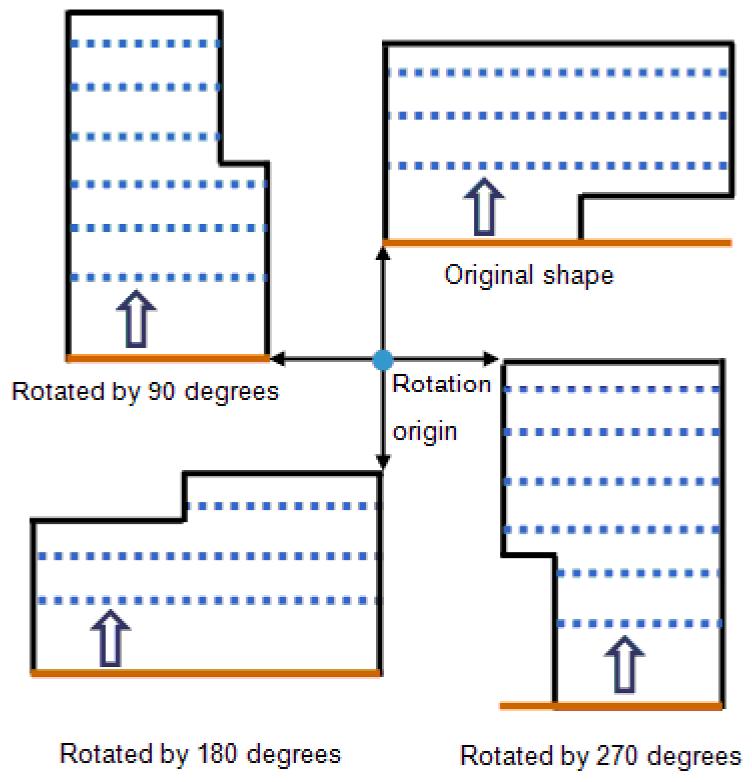
The following figures illustrate the various attributes of a snap pattern:



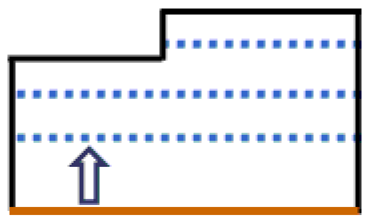
Vertical Direction



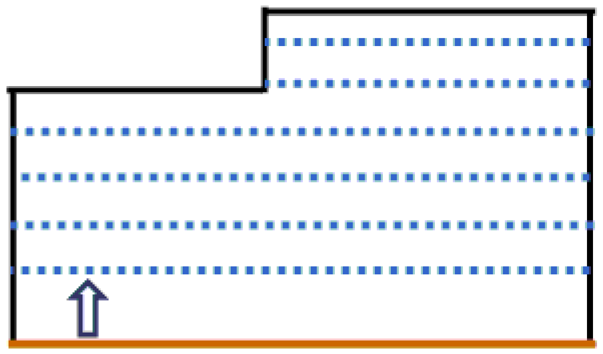
Horizontal Direction


Original shape
Rotated by 90 degrees
Rotation
origin
Rotated by 180 degrees
Rotated by 270 degrees

Rotation



Original shape



Magnified Shape

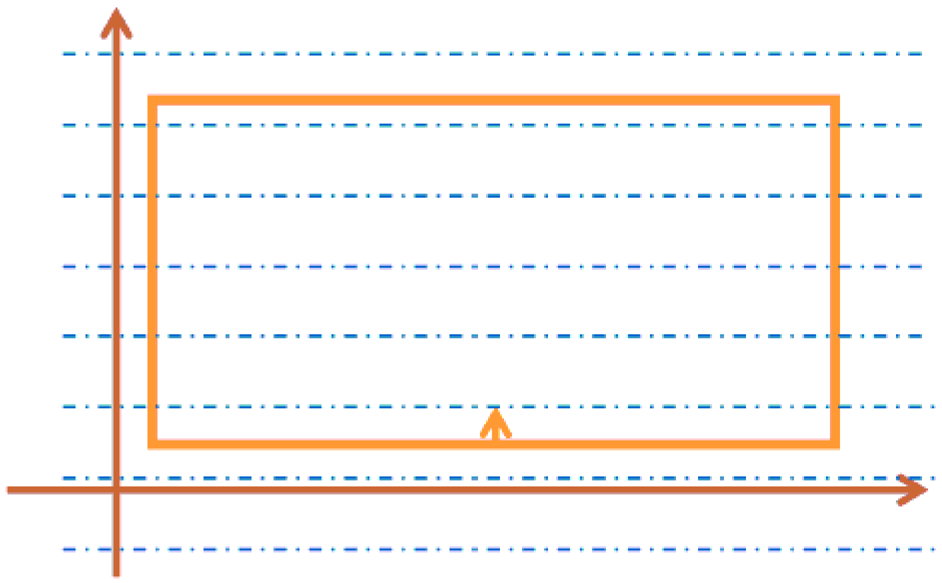
Activating a Global Snap Pattern Definition

A global snap pattern definition applies to the entire cellview. It can be activated through a:

- Snap Pattern Shape
- Global snapPatternDef

Snap Pattern Shape

A global snap pattern definition is activated when a snap pattern shape is drawn on an LPP with a global snap pattern definition. In the figure below, the snap pattern shape is shown in orange. The grid offset, indicated by the arrow, is relative to the bottom edge (for vertical direction) or the left edge (for horizontal direction) of the snap pattern shape.

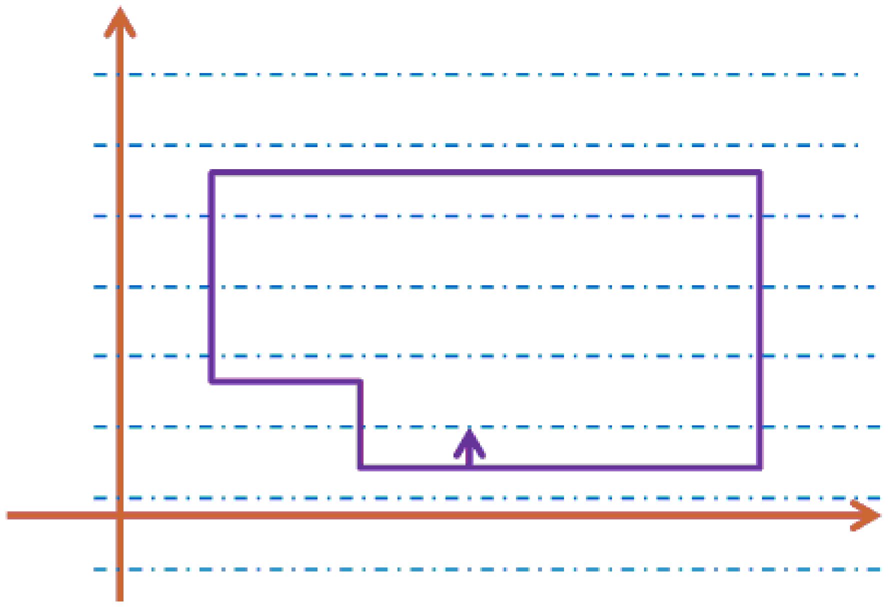


Global snap pattern grid anchored by snap pattern shape

Global snapPatternDef

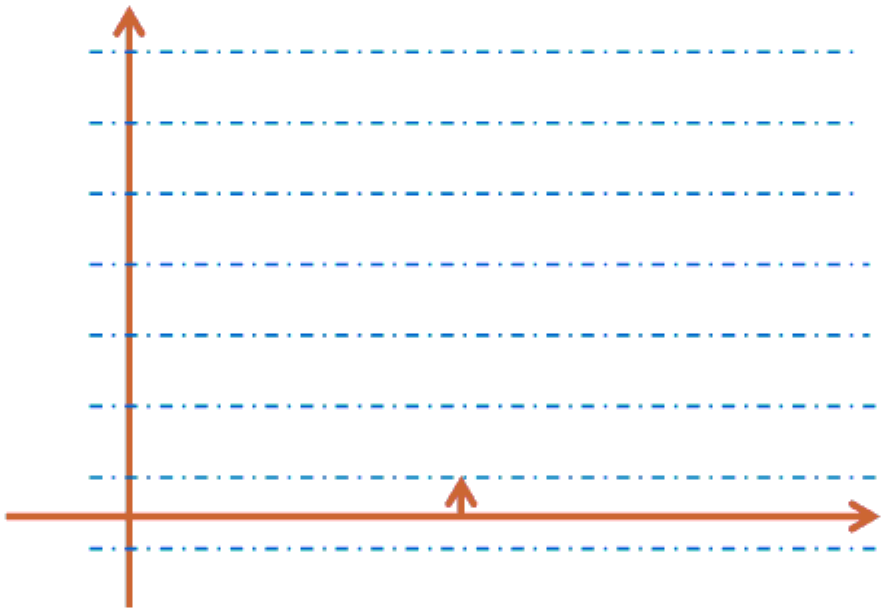
When the snap pattern definition is enabled through the snapGridHorizontal or snapGridVertical constraint, the snap pattern shape is not necessarily required. If a snap boundary exists and no shape is drawn on a global grid LPP, the grid is anchored to the snap boundary. The offset is relative to the lower-left edge of the snap boundary. The grid extends into infinity on all sides. If the snap boundary does not exist and no shape is drawn on a global grid LPP, the grid is anchored to the PRBoundary, shown in purple in the following figure.

For more information about how to control the anchoring of global snap pattern definitions, see Working with the Track Pattern Assistant in *Virtuoso® Width Spacing Patterns User Guide*.



Global snap pattern grid anchored by PRBoundary

If neither a snap boundary nor a PRBoundary exists and no shape is drawn on a global grid LPP, the grid is anchored to the origin axis (X-axis for the vertical direction, Y-axis for the horizontal direction), as shown in the following figure:

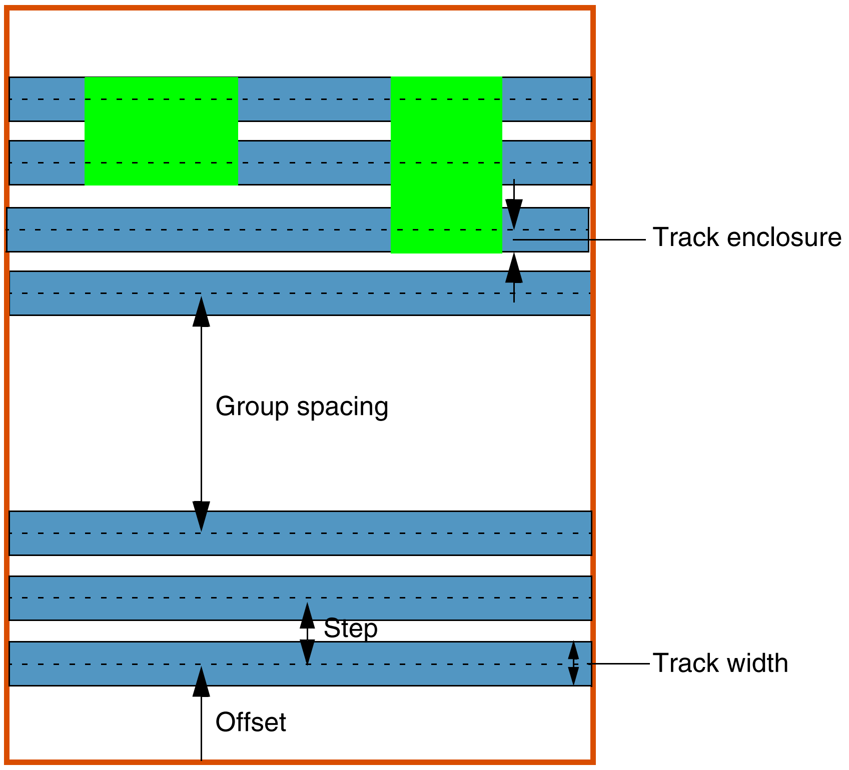


Global snap pattern grid anchored by origin axis

Track Groups

(ICADVM20.1 Only – 95511) This feature lets you create track groups with variable spacing. Additionally, each track group in the snap pattern definition can contain a different number of tracks. For example, in the figure below, the bottom track group has three tracks and the top track group has four tracks. The track enclosure is set to half the track width, resulting in snap pattern shapes (in green) that enclose the track width.

This example also shows how track groups are positioned. The distance from the reference edge to the centerline of the first track is equal to the snap pattern definition offset, and each group has its own spacing that specifies the distance between the last track in the current group and the first track in the next group.



Track groups with variable group spacing

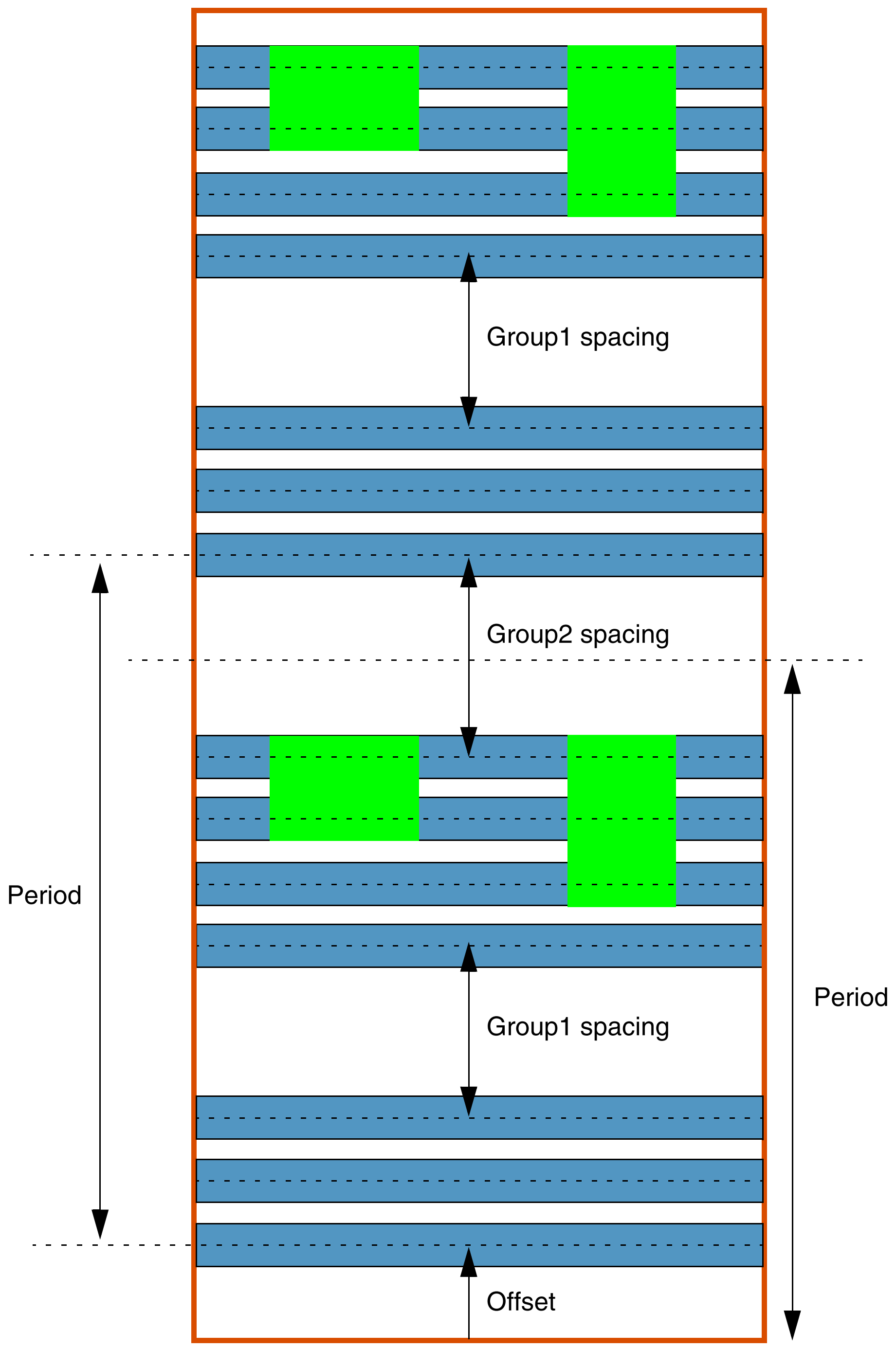
Within each group, the `'step` attribute defines track-to-track spacing, and the `'trackWidth` attribute defines the width of each track. Snap pattern shapes enclose tracks by using the `'enclosures` attribute.

Track groups defined in a snap pattern definition are repeated to fill snap pattern shapes of arbitrary sizes. The period in this case is defined as the total height of one repetition of the full snap pattern definition and is calculated as follows:

Track group height, G_T = *`groupSpace`* + (*`trackCount`* - 1) * *`step`*

Period = Sum of all G_T

For example, in the figure below, you can see two repetitions of a snap pattern with two track groups. The spacing of the last track group is used when a snap pattern is tall enough to accommodate a repetition of the pattern, that is, the spacing of the last track group is the distance from the last track in the last track group to the first track in the first track group.



Repeating track groups

Arguments

`t_name`	The name of the snap pattern definition. The snap pattern definition must have a unique name.
`tx_layer`	The layer to which the snap pattern definition applies. Valid values: The layer name or the layer number
`tx_purpose`	The purpose to which this snap pattern definition applies. The parent purpose of *`tx_purpose`* must be `annotation`. Valid values: The purpose name or the purpose number
`'step`	The spacing between snap pattern tracks. This value must be greater than zero.
`'stepDirection`	The direction in which snap pattern tracks are created. When the direction is `vertical`, track spacing is applied from bottom to top. When the direction is `horizontal`, track spacing is applied from left to right.
`'type`	The snap pattern definition type. ■ `local`: A local snap pattern definition is active only if a corresponding snap pattern shape is drawn in the layout. ■ `global`: It applies to the entire cellview and can be activated in two different ways: ❑ By drawing a corresponding snap pattern shape ❑ By enabling it through a constraint When snapping a shape to a snap pattern grid, snap pattern shapes whose snap pattern definition is of type `local` have higher priority than those of type `global`. The default is `local`.
`'offset`	The distance of the first snap pattern track from the bottom (for the vertical direction) or the left (for horizontal direction) edge of the bbox shape. The offset must be greater than or equal to zero.
`'snappingLayers`	A list of layers, purposes, and enclosures that determine the shapes that snap to the snap pattern.

`'layer`	A list of layers. Shapes on these layers snap to the tracks of this snap pattern. The list is specified in this format: `'layer ( `*`tx_snapLayer`*` … )`
`'enclosures`	A list of enclosures of a snapping shape beyond a grid line in the snap pattern direction. If vias and top-level shapes are present on the snapping layer (and snapping purpose, if used), they are snapped to the snap pattern grid by using the `'enclosures` value. The list is specified in this format: `'enclosures ( `*`f_enclosure`*` … )`
`'purposes`	A list of purposes. Shapes on `'snappingLayers` with a specific purpose snap to a snap pattern. `'layer` applies to all purposes if `'purposes` is not specified. The list is specified in this format: `’purposes ( `*`tx_snapPurpose`*` … )`

`'snappingMode`	The snapping mode. The following values are supported: ■ `multiTrackCenter`: Default value used when the attribute is not specified. It indicates if a shape can be centered on any number of tracks in a way that when one of the `'enclosure` values is met on both sides, it is snapped to the grid. When a shape is created, it is assumed that all edges will be on the grid and the first and second points always snap to the grid. If the combined snapping layer BBox inside an instance can be centered in a way that the `'enclosure` value is met on both sides, then the instance snaps to the grid using that snapping layer. If no snapping layer BBox inside an instance snaps to the grid, then the instance is not snapped. ■ `singleTrackCenter`: Indicates if a shape can be centered on a single track in a way that when the `'enclosure` value is met on both sides, it is snapped to the grid. When a shape is created, it is not assumed that all edges will be on the grid. Only a shape with dimension that is twice the `'enclosure` value would snap to the grid. In this case, the combined snapping layer BBox is not considered. If the device contains any shape meeting the dimension rule, the shape is snapped even if the combined layer BBox does not meet the enclosure on both sides.
`'exceptOverlap` *`l_overlapLPPs`*	A list of layer-purpose name pairs. Applicable only when `'type` is `global`. Snapping does not happen if a shape is completely inside the shape with one of the LPPs specified using `'exceptOverlap`. If the shapes overlap partially, snapping does take place. The list is specified in this format: `((`*`t_layerName t_purposeName`*`)…)`
`'trackWidth`	The width of the physical shape that each track represents. This value is used only for display and illustration purposes. The default value is zero.
`'trackGroups`	(ICADVM20.1 Only – 95511) Groups of tracks with variable spacing. When `'trackGroups` is used, at least one count-space pair must be specified.

`'count`	(ICADVM20.1 Only – 95511) The number of tracks in each track group.
`'space`	(ICADVM20.1 Only – 95511) The spacing to the next track group.

Example

```
layerDefinitions(
        techPurposes(
            (fb       1001 fb   'parent annotation)
            (fbdummy  1002 fbd  'parent annotation)
        ) ;techPurposes
) ;layerDefinitions
layerRules(
        snapPatternDefs(
            (finfet     ("FF" "fb")
            'offset           0.007
            'type             "local"
            'step             0.048
            'stepDirection    "vertical"
            'snappingLayers   (
                ('layer "Active" 'enclosures (0.007)
                'snappingMode "singleTrackCenter"
                'exceptOverlap ("GG" "Purp1")
                )
            )
            'trackWidth       0.014
            )
            (finfetDummy ("FF" "fbdummy")
            'offset           0.007
            'type             "local"
            'step             0.048
            'stepDirection    "vertical"
            'snappingLayers   (
                ('layer "Active"
                'enclosures (0.007)
                'purposes ("dummy" "dummy1")
                )
            )
            'trackWidth        0.014
            )
    ) ;snapPatterns
) ;layerRules
```

Related Topics

- layerRules

relatedSnapPatterns

```
layerRules(
     relatedSnapPatterns(
          ( t_name
               'snapPatternDefs (
                    (t_snapPatternDefName
                         ['patterns       (l_patternNames)]
                         ['patternGroups  (l_patternNames)]
                    ) …)
          ['extraLPP (tx_layer tx_purpose)]
          ['regionSnapPitchHorizontal f_snapH]
          ['regionSnapPitchVertical f_snapV]
          )
     ) ;relatedSnapPatterns
) ;layerRules
```

Description

(ICADVM20.1 Only – 95511) Specifies predefined groupings of `snapPatternDefs` and `widthSpacingSnapPatternDefs` used to create groups of regions in the layout. The allowed patterns and pattern groups on each region in the group are initialized to the values specified by the `'patterns` and `'patternGroups` attributes. The active pattern of the region is set to the first pattern in the `'patterns` attribute list.

Patterns and groups are specified only for `widthSpacingSnapPatternDefs`. These attributes are not used for `snapPatternDefs`.

Arguments

t_name	The name of the `relatedSnapPattern` definition.
`'snapPatternDefs`	The snap pattern definition.
t_snapPatternDefName	The name of the snap pattern definition. This name can refer to either a `snapPatternDef` or a `widthSpacingSnapPatternDef`.
`'patterns (`*l_patternNames*`)`	A list of allowed width spacing pattern names for `widthSpacingSnapPatternDefs`. This is used as the initial set of allowed patterns on regions created using this `relatedSnapPattern` definition.

`'patternsGroups (`*`l_patternNames`*`)`

A list of allowed width spacing pattern group names for `widthSpacingSnapPatternDefs`. This is used as the initial set of allowed pattern groups on regions created using this `relatedSnapPattern` definition.

`'extraLPP (`*`tx_layer tx_purpose`*`)`

An additional layer-purpose pair. A shape on the LPP is created automatically when an instance of the related snap pattern is placed in the layout.

`'regionSnapPitchHorizontal` *`f_snapH`*

The pitch at which horizontal region edges snap in the vertical direction. This value overrides the X snapping grid values.

The default value is 0.

`'regionSnapPitchVertical` *`f_snapV`*

The pitch at which vertical region edges snap in the horizontal direction. This value overrides the Y snapping grid values.

The default value is 0.

Example

```
relatedSnapPatterns(
    ("1XGroup"
        'snapPatternDefs (
            ("fb1")
            ("m1_grid" 'patterns ("1X" "1X-4X"))
            ("m2_grid" 'patterns ("1X"))
            ("m3_grid" 'patterns ("1X-3X" "1X-4X"))
        )
        'extraLPP ("Active" "drawing")
        'regionSnapPitchHorizontal 0.1
        'regionSnapPitchVertical 0.2
    )
) ;relatedSnapPatterns
```

Specifies a `relatedSnapPatterns` definition named `1XGroup`. It contains entries for four grids: the fin grid, `fb1`, and the routing grid for `m1`, `m2`, and `m3`. When a region group is created using this definition, it would contain four shapes—one for each grid. The allowed patterns on the `m1_grid` region would be set to `1X` and `1X` - `4X`. An additional LPP is specified along with related pitch values.

Related Topics

- layerRules

widthSpacingPatterns

```
layerRules(
     widthSpacingPatterns(
       (t_name
         ['offset g_offset ['repeatOffset]]
         ['startingColor g_maskColors | 'shiftColor]
         ['allowedRepeatMode {"any" | "none" | "steppedOnly" | "flippedOnly"}
           ['defaultRepeatMode
             {"stepped" | "flippedStartsWithOdd" | "flippedStartsWithEven"}
           ]
         ]
         'pattern(
           (['repeat g_repeat]
            ['wireTypes (l_wireTypes)]
            ['colors (g_colors …)]
            ['displayPackets (l_packets)]
            'spec(('width g_width
                 'space g_space
                 ['wireType t_wireType]
                 ['color g_maskColors]
                 ['displayPacket t_packet]
                ) …
            ) ;spec
           ) …
         ) ;pattern
       )
     ) ;widthSpacingPatterns
) ;layerRules
```

where,

```
g_colors := "mask1Color" | "mask2Color" | "mask3Color" | "grayColor"

g_maskColors := "mask1Color" | "mask2Color" | "mask3Color"
```

Description

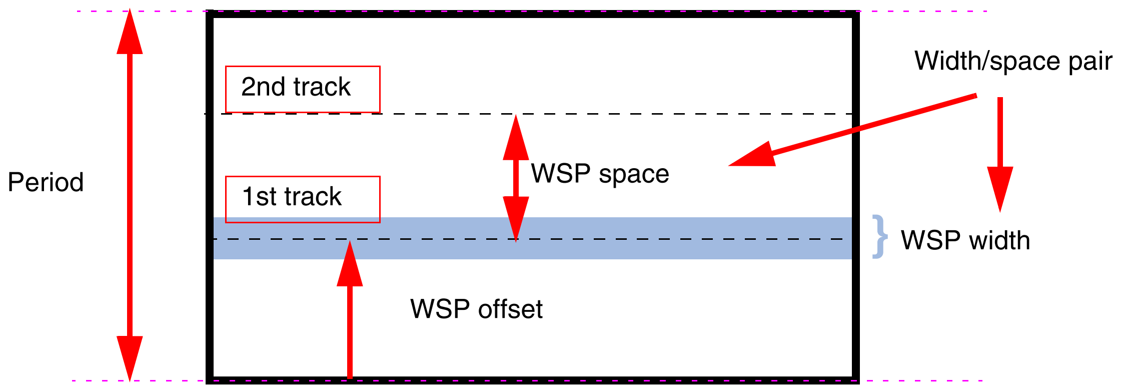
(ICADVM20.1 Only – 95511) Defines the tracks on which shapes can be placed. It also defines the width of shapes on those tracks. Each pattern may span one or multiple periods. A `widthSpacingPattern` (WSP) by itself has no layer and no direction. It defines the width, spacing, and, optionally, the color and the wire type for a set of tracks. The pattern is then applied to a routing layer in a particular direction by specifying it as the active pattern in a `widthSpacingSnapPatternDef`.

Each track can have an optional wire type, which is a string label that can be used to refer to tracks by a name. During layout editing, tracks can be enabled or disabled through wire type filtering.

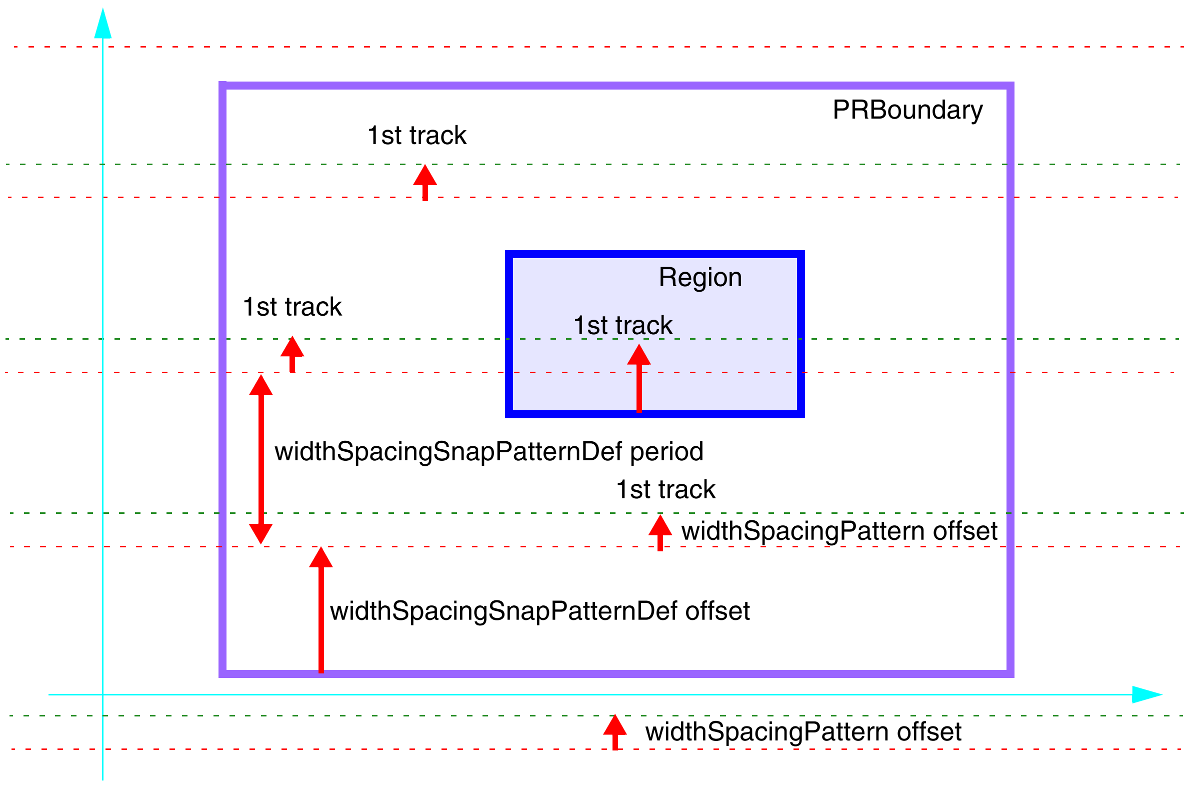
There are two ways to specify the color of each track in a pattern. In most cases, tracks have alternating colors and no adjacent pair of tracks has the same color. This can be specified by using the `'startingColor` attribute. This attribute specifies the color of the first track; all other tracks are colored automatically with alternating colors.

In certain situations, not all tracks are colored or adjacent tracks need to be assigned the same color. This can be done by using the `'color` attribute of each individual track or by using the `'colors` attribute to specify a list of colors for a set of tracks. In addition, the `'shiftColor` attribute can be used to specify whether all track colors are shifted when the pattern repeats, or whether each repetition of the pattern has the same color assignment.

As shown in the figure below, each `widthSpacingPattern` is a sequence of width-space pairs. The offset indicates the location of the first track from the period grid.



A collative view of the `widthSpacingSnapPatternDef` offset and the `widthSpacingPattern` offset is illustrated in the figure below:


PRBoundary
1st track
Region
1st track
1st track
widthSpacingSnapPatternDef period
1st track
widthSpacingPattern offset
widthSpacingSnapPatternDef offset
widthSpacingPattern offset

The red dotted lines indicate the global period grid in each stripe of the period grid. This grid is anchored to the PRBoundary by using the offset of the `widthSpacingSnapPatternDef`.

The green dotted lines indicate the first track in each period. Outside pattern regions, the active global default pattern is applied and its offset specifies the distance from the period grid line to the first wiring track.

Finally, a region shape is used to apply a non-default active pattern in an area of the cellview. The active pattern specified on the region overrides the default active pattern and its offset specifies the location of the first track.

Repeat Modes and Pattern Flipping

Repeat modes determine how a pattern is interpreted for use in adjacent periods. The allowed and default repeat modes are specified on a `widthSpacingPattern`; the repeat mode can be specified in the layout on global grids and pattern regions.

The `'allowedRepeatMode` attribute indicates how a pattern is allowed to repeat when a region stretches across more than one period, and the `'defaultRepeatMode` attribute is used to initialize the repeat mode on regions and global grids when they are first created.

The following are valid combinations for allowed repeat mode (`'allowedRepeatMode`) and default repeat mode (`'defaultRepeatMode`) attributes:

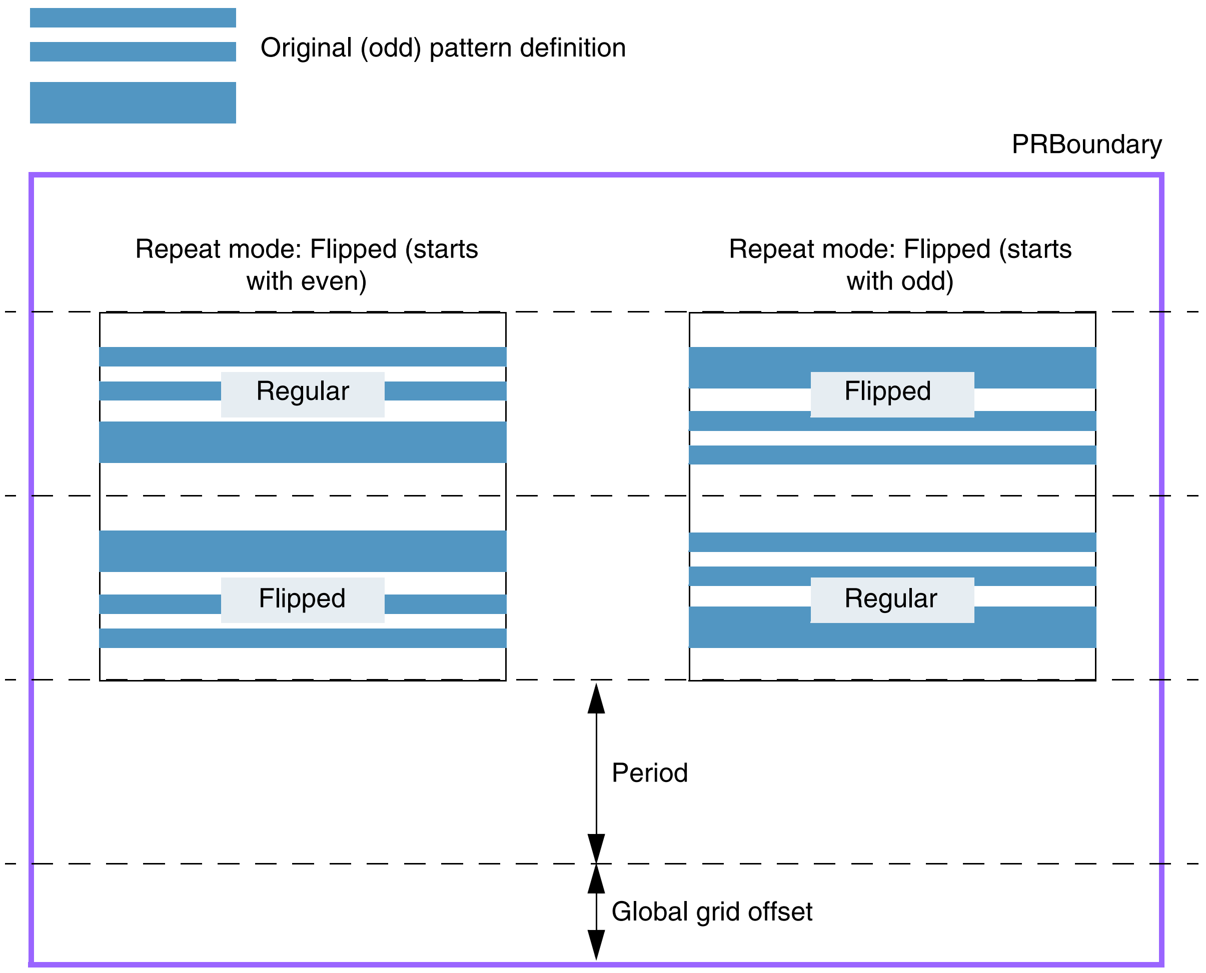
Allowed repeat mode	Default repeat mode
`any`	`stepped`
	`flippedStartsWithOdd`
	`flippedStartsWithEven`
`steppedOnly`	`stepped`
`flippedOnly`	`flippedStartsWithOdd`
	`flippedStartsWithEven`

In a layout:

- A repeat mode can be defined at the following two levels:
  - On a region: A region has a set of allowed patterns and a repeat mode can be specified for each pattern.
  - On a global grid: Each global grid also has a set of allowed patterns and a repeat mode can be specified for each pattern. A global grid in a cellview is identified by a global `widthSpacingSnapPatternDef` (the one that is set as active in the Track Pattern Assistant).
- A repeat mode defined on a global grid or a pattern region can be of one of the following types:
  - `stepped`: The pattern is the same in every period.
  - `flippedStartsWithOdd`: The pattern is flipped in every other period. The first period is not flipped.
  - `flippedStartsWithEven`: The pattern is flipped in every other period. The first period is flipped.

A repeat mode is not inherited and does not have a look-up precedence. As a result, when the repeat mode of the global grid changes, the regions on that global grid are not affected. However, when a region with a pattern for which the repeat mode is set is copied, the repeat mode is also copied to the newly created region.

The following example illustrates how a pattern in a region appears when "even" and "odd" flipped repeat modes are applied to the pattern:

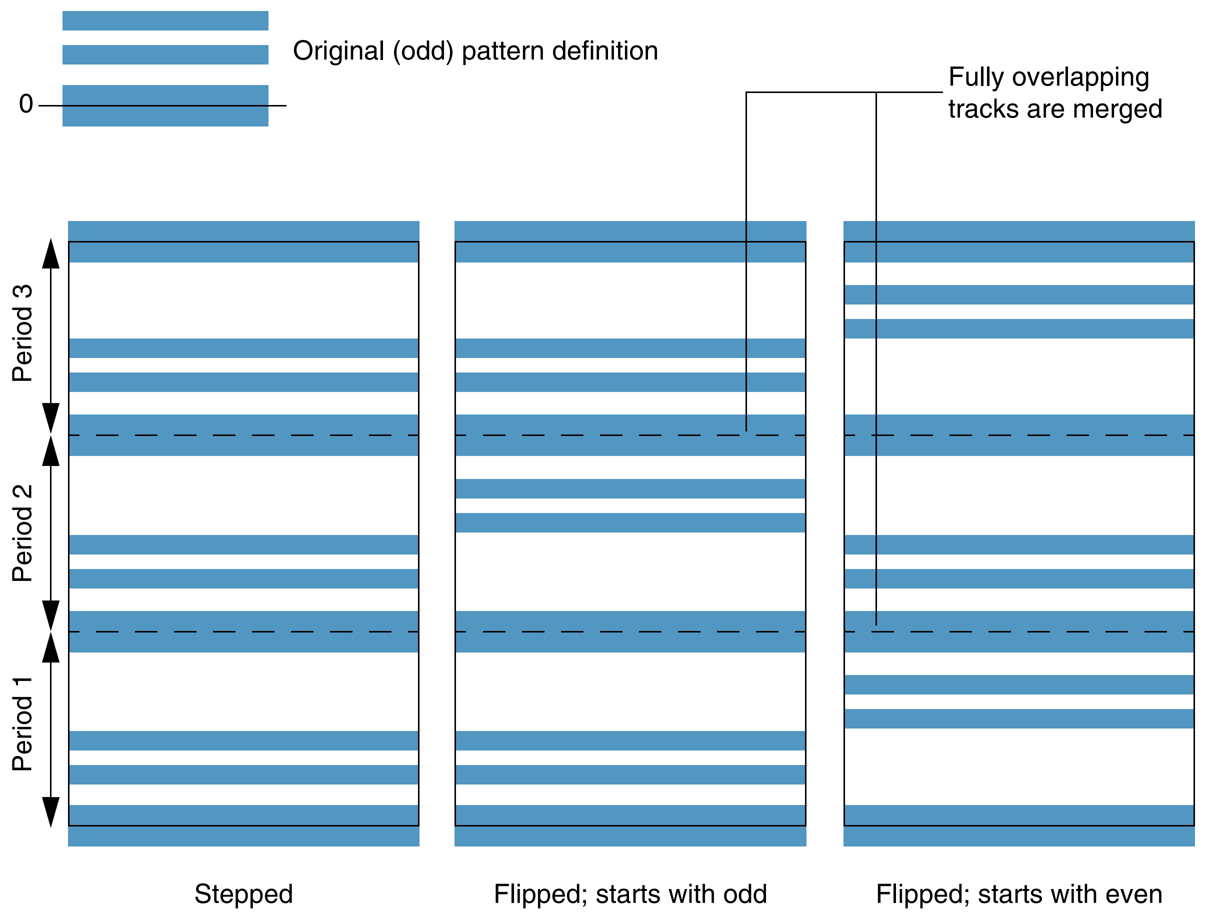

Original (odd) pattern definition
PRBoundary
Repeat mode: Flipped (starts with even)
Repeat mode: Flipped (starts with odd)
Regular
Flipped
Flipped
Regular
Period
Global grid offset

Note: When a starting color is assigned to the pattern, the starting color always applies to the bottommost track in the pattern, regardless of whether the pattern is flipped or not. This ensures tracks with alternating colors across the cellview.

Patterns with Zero Offset

When a pattern has zero offset, the centerline of the first track is located on the period line or the bottom edge of the region. In a legal pattern, the height of the pattern needs to be compatible with the period and the height of the region. Therefore, a track at offset zero is also present on the top period line or the top region boundary. When a pattern with zero offset is flipped, the top and bottom tracks overlap and are merged into one single track.

The following example illustrates how a pattern with zero offset appears when "even" and "odd" flipped repeat modes are applied to the pattern in a region:


Original (odd) pattern definition
0
Fully overlapping tracks are merged
Period 3
Period 2
Period 1
Stepped
Flipped; starts with odd
Flipped; starts with even

Arguments

`t_name`	The name of the width spacing pattern.
`'offset` *`g_offset`*	The distance of the center line of first track from the period track.
`'repeatOffset`	A flag to indicate whether the offset is applied when a pattern is repeated.
`'startingColor` *`g_maskColors`*	The color—`mask1`, `mask2`, or `mask3`—of the first track. When the pattern is repeated and the number of tracks is odd, track colors are shifted automatically on repeat. When this attribute is used, all tracks are automatically colored with alternating colors. This attribute is optional, the default value is gray (uncolored).
`'shiftColor`	A flag to indicate whether track colors are shifted when a pattern is repeated. This attribute cannot be used when `'startingColor` is used.
`'allowedRepeatMode`	This indicates how a pattern repeats when it stretches across more than one period. Valid values: ■ `any` (default): No restrictions on how the pattern repeats. The pattern can be stepped and flipped. ■ `none`: The pattern is not allowed to repeat; only a single period can be used. ■ `steppedOnly`: The pattern must be stepped. ■ `flippedOnly`: The pattern must be flipped in alternate periods; the pattern cannot be stepped.

`'defaultRepeatMode`	This is used by applications to initialize the repeat mode on regions and global grids when they are first created. Valid values include: ■ `stepped`: This can be specified only when `'allowedRepeatMode` is set to `any` or `steppedOnly`. ■ `flippedStartsWithOdd`: This can be specified only when `'allowedRepeatMode` is set to `any` or `flippedOnly`. ■ `flippedStartsWithEven`: This can be specified only when `'allowedRepeatMode` is set to `any` or `flippedOnly`.
`'repeat` *`g_repeat`*	The number of repeats for the track group. This attribute should be greater than 1.
`'wireTypes (`*`l_wireTypes`*`)`	
	A list of wire types for the track group (only if *`d_repeat`* is specified). An empty string in the list can be used if you do not want to specify a wire type for a track.
`'colors (`*`g_colors`*`)`	
	A list of track colors for the track group (only if *`d_repeat`* is specified). The `grayColor` entry can be used to not specify a color for a track. This attribute would not be allowed if a starting color is specified.
`'displayPackets (`*`l_packets`*`)`	
	A list of display packets for the track group (only if *`d_repeat`* is specified). An empty string in the list can be used if you do not want to specify a display packet for a track.
`'width` *`g_width`*	The width of shapes on this track.
`'space` *`g_space`*	The center-line distance of the next track from the current track, in the period direction.
`'wireType` *`t_wireType`*	

A string that represents the wire type. To be specified only if `'colors` and `'wireTypes` are not used.

`'color` *`g_maskColors`*

The color—`mask1Color`, `mask2Color`, or `mask3Color`—of the specified track. To be specified only if `'colors` and `'wireTypes` are not used.

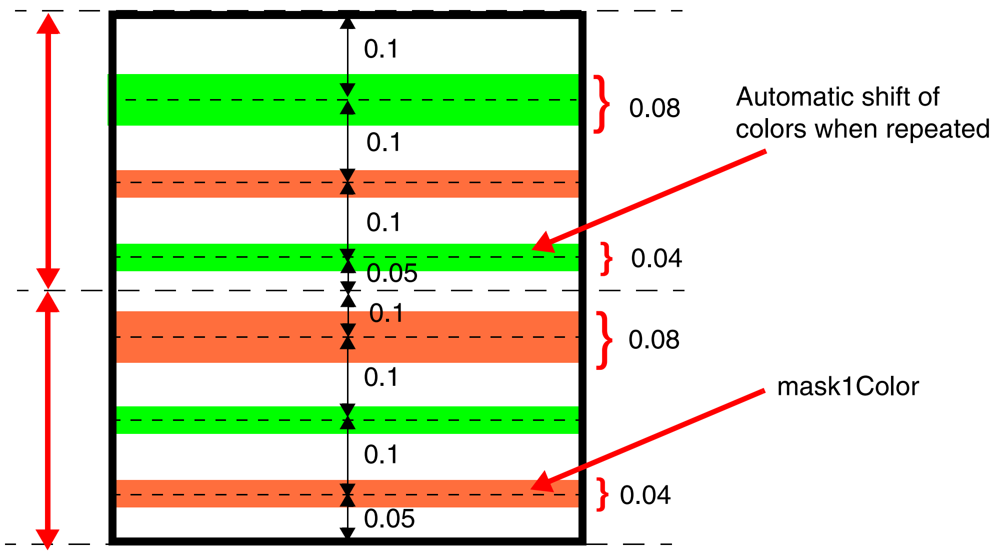
`'displayPacket` *`t_packet`*

A string that represents the display packet name. To be specified only if `'colors`, `'wireTypes`, and `'displayPackets` are not used.

Example 1

```
layerRules(
     widthSpacingPatterns(
          ("2-1x-1-2x"
               'offset 0.05
               'repeatOffset
               'startingColor "mask1Color"
               'pattern(
                    (
                         'repeat 2
                         'spec (('width 0.04 'space 0.1))
                    )
                    (
                         'spec (('width 0.08 'space 0.1))
                    )
               )
          )
     ) ;widthSpacingPatterns

) ;layerRules
```

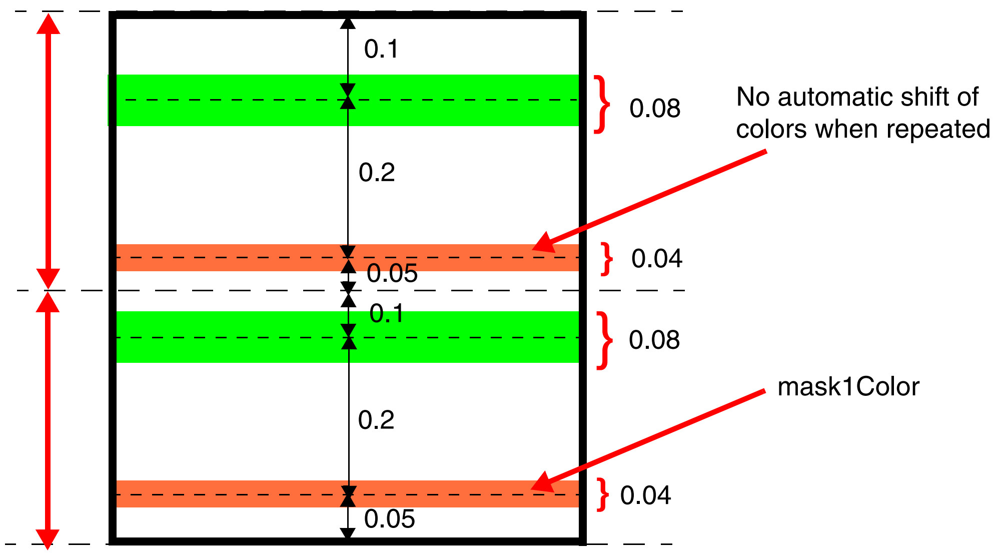
Specifies that the tracks are alternately colored and are shifted on repeat because the number of tracks in the pattern is odd.



Example 2

```
layerRules(
    widthSpacingPatterns(
        ("1-1x-1-2x"
            'offset 0.05
            'repeatOffset
            'startingColor "mask1Color"
            'pattern (
                (
                    'repeat 2
                    'spec (
                        ('width 0.04 'space 0.2)
                        ('width 0.08 'space 0.1)
                    )
                )
            )
        )
    )
) ;layerRules
```

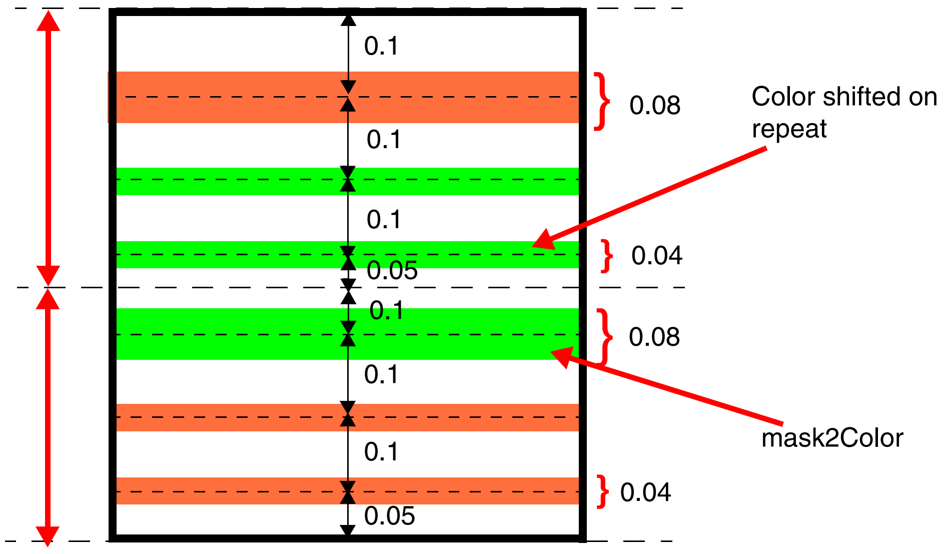
Specifies that the tracks are alternately colored, but are not shifted on repeat because the number of tracks in the pattern is even.



Example 3

```
layerRules(
    widthSpacingPatterns(
        ("2-1x-1-2x"
            'shiftColor
            'offset 0.05
            'repeatOffset
            'pattern (
                (
                    'repeat 2
                    'colors ("mask1Color" "mask1Color")
                    'spec (('width 0.04 'space 0.1))
                )
                (
                    'spec (('width 0.08 'space 0.1 'color "mask2Color"))
                )
            )
        )
    )
) ;layerRules
```

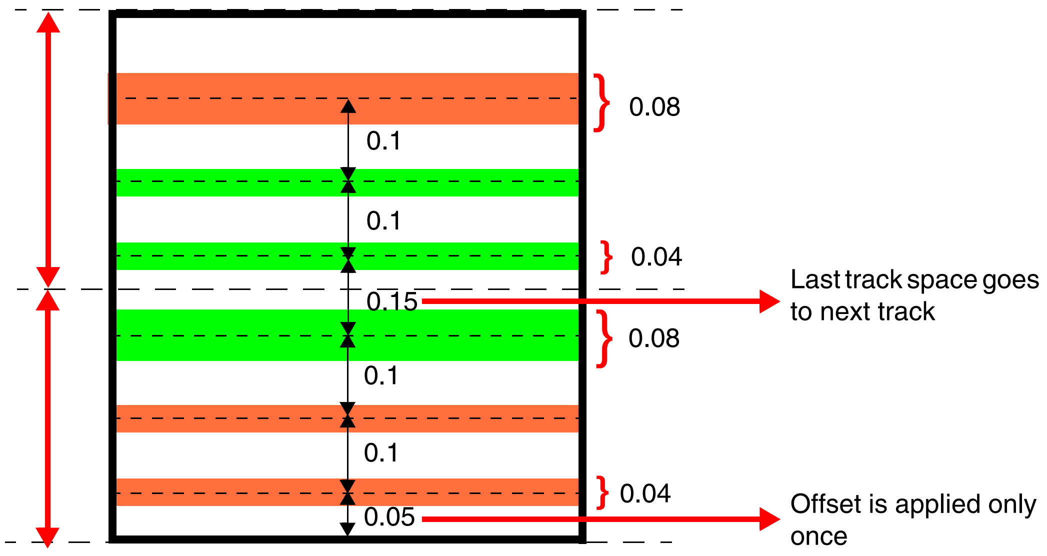
Specifies that the tracks are colored individually and some adjacent tracks have the same color. When the pattern is repeated, the colors are shifted because the `'shiftColor` attribute is used.



Example 4

```
layerRules(
    widthSpacingPatterns(
        ("2-1x-1-2x"
            'shiftColor
            'offset 0.05
            'pattern (
                (
                    'repeat 2
                    'colors ("mask1Color" "mask1Color")
                    'spec (('width 0.04 'space 0.1))
                )
                (
                    'spec (('width 0.08 'space 0.15 'color "mask2Color"))
                )
            )
        )
    )
) ;layerRules
```

Specifies that the offset is applied only once because `'repeatOffset` is not given. The spacing specified for the last track in the pattern goes to the next track, as shown.


0.08
0.1
0.1
0.04
0.15
Last track space goes to next track
0.08
0.1
0.1
0.04
0.05
Offset is applied only once

Related Topics

layerRules

widthSpacingPatternGroups

```
layerRules(
     widthSpacingPatternGroups(
          (t_name
               'members (l_patternNames)
          ) …
     ) ;widthSpacingPatternGroups
) ;layerRules
```

Description

(ICADVM20.1 Only – 95511) Defines groups of widthSpacingPatterns specified in the technology file. Such a group is a convenient way to refer to a set of patterns by name. Groups can be used in a set of allowed patterns on a `widthSpacingSnapPatternDef` and region shapes.

Arguments

`t_name`	The name of the `widthSpacingPatternGroup`.
`'members (`*`l_patternNames`*`)`	The list of width spacing patterns names that are members of this group.

Example

```
layerRules(
    widthSpacingPatternGroups(
        ( "basic"
            'members ( "minWidth" "2XWidth")
        )
        ( "multiWSP"
            'members ( "WSP1" "WSP2" )
        )
    )
) ;layerRules
```

Defines pattern groups `basic` and `multiWSP`. Each group has two patterns as members.

Related Topics

layerRules

widthSpacingSnapPatternDefs

```
layerRules(
     widthSpacingSnapPatternDefs(
          (t_name (tx_layer tx_purpose) | tx_layer
               'period g_period
               'direction "vertical" | "horizontal"
               ['offset g_offset]
               ['snappingLayers(
                    ('layer tx_snapLayer  ['purposes l_purposes])
                    …
                )
               ]
               {['patterns (l_patternNames)]
                ['patternGroups (l_patternGroupNames)]
               }
               'defaultActive t_defaultActivePatternName
               ['orthogonalGrid g_gridType]
          )
     ) ;widthSpacingSnapPatternDefs
) ;layerRules
```

Description

(ICADVM20.1 Only – 95511) Defines a coarse-grain grid called the period grid in a cellview. The available wiring tracks are specified between the period grid lines using width spacing patterns. By default, the default active pattern applies in every period stripe. Rectangles and polygons on the `widthSpacingSnapPatternDef`'s LPP can be used to customize the width spacing patterns in a particular area of the cellview. Such rectangles and polygons are referred to as "regions".

`widthSpacingSnapPatternDefs` also define a set of allowed patterns and pattern groups. All shapes on the snapping layers must conform to one of these allowed patterns. Regions can be used to override the default set of allowed patterns and pattern groups in certain areas of the cellview.

The snapping layers are the layers to which the `widthSpacingSnapPatternDef` applies. Typically, one `widthSpacingSnapPatternDef` is used per routing layer.

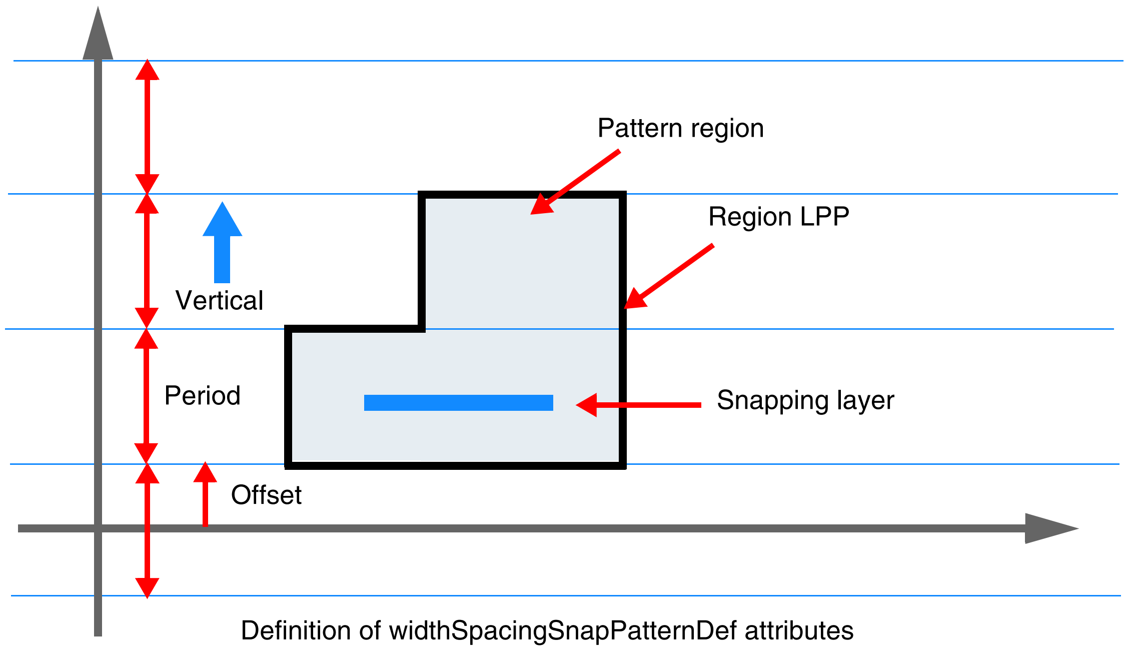
The direction of a `widthSpacingSnapPatternDef` is the direction in which the spacing between period grid lines is measured, that is, a vertical `widthSpacingSnapPatternDef` would have horizontal period lines.

The global period grid defined by a `widthSpacingSnapPatternDef` is anchored to the lower edge, along with an optional offset, of the SnapBoundary or PRBoundary. If a

SnapBoundary or PRBoundary does not exist, the global grid is anchored to the origin axis. This behavior can be disabled using the `dbSetGlobalGridReferenceType` function.

A `widthSpacingSnapPatternDef` applies to a cellview only if it is enabled through a `snapGridVertical` or `snapGridHorizontal` constraint. This constraint can be specified library-wide in the `foundry` constraint group or in a cellview's default constraint group.

The following figure illustrates the coarse-grain period grid with vertical direction, anchored to the origin axis.



Definition of widthSpacingSnapPatternDef attributes

Arguments

`t_name` — The name of the `widthSpacingSnapPatternDef`. The name must be unique.

`(tx_layer tx_purpose) | tx_layer`

The LPP on which pattern regions are drawn. The LPP cannot be used with a `snapPatternDef`.

If only *`tx_layer`* is specified, the `track` purpose is used.

`'period g_period` — The spacing between coarse-grain period tracks.

`'direction "vertical" | "horizontal"`

The direction in which the period spacing is applied.

`'offset g_offset` — The distance of the nearest period track to the anchor reference (either the lower edge of the SnapBoundary, PRBoundary, or the origin axis).

The default offset is zero.

`'snappingLayers ('layer tx_snapLayer ['purposes l_purposes])`

A list of layers, and optionally purposes, to which this `widthSpacingSnapPatternDef` applies.

If a list of purposes is not specified, then the `snappingLayer` applies to all purposes.

`'patterns l_patternNames`

The default list of allowed patterns. This can be customized in the local areas of the cellview by creating region shapes.

`'patternGroups l_patternGroupNames`

A list of allowed pattern groups. The union of all of these groups, as well as the patterns specified by the `'patterns` attribute, is the default set of all legal patterns.

`'defaultActive t_defaultActivePatternName`

The default pattern shown in the layout in areas where no region is drawn.

`'orthogonalGrid g_gridType`

The WSP grid to use for line-ends. The following values are valid:

- `upperLower`: Applies to both ends of the wires.
- `lower`: Applies to the lower line-ends of vertical wires and the left line-ends of horizontal wires.
- `upper`: Applies to the upper line-ends of vertical wires and the right line-ends of horizontal wires.
- `nil`: The default value, which is used when this argument is not specified. In this case, the line-end grid is not configured for the `widthSpacingSnapPatternDef` object.

Example

```
widthSpacingSnapPatternDefs(
    ( M2WSP
        ("Metal2"        "localWSP")
        'period         0.768
        'direction      "vertical"
        'offset         0
        'snappingLayers ( ('layer Metal2) )
        'patterns       ("stdCell")
        'patternGroups  ("basic" "multiWSP")
        'defaultActive  "minWidth"
    )
)
```

Defines a `widthSpacingSnapPatternDef` named `M2WSP` that specifies a vertical period grid in which each stripe is 0.768 high. The first period grid line falls on the lower edge of the PRBoundary, or the origin axis, because the offset is zero.

This `widthSpacingSnapPatternDef` applies to the `Metal2` routing layer. By default, the `minWidth` pattern is shown in each stripe. The default set of allowed patterns includes all patterns in the `basic` and `multiWSP` groups, as well as the `stdCell` pattern. The active and allowed patterns can be customized by creating region shapes on the `Metal2/localWSP` LPP.

Related Topics

layerRules

Current Density and Current Density Tables

Current density tables allow conditional evaluation and application of a layer attribute value by means of lookup tables. When specifying current density tables, you must follow the syntax indicated. Current density tables always apply to one layer and can be one- or two-dimensional, with the attribute value being determined by one or two other design values. All `currentDensityTable` attributes must be specified as indicated in their syntax statements.

The table specification (*`l_table`*) always takes the following syntax:

For a 1-D table	`(` *`g_index g_value …`* `)`
For a 2-D table	`(` *`g_index1 g_index2 g_value …`* `)`

where, an index is a value for the design value being used to determine the attribute value.

- The table entries must be specified in the ascending order.
- All table entries must be explicitly specified; otherwise, the technology file compiler uses the default value for the unspecified entries. If no default value has been specified in the `currentDensityTables` section, the compiler fills in a `0` for the missing entries in the table.
- Each table entry is read as greater than or equal to. For example, consider the following table entries:

```
currentDensityTables(
    ( peakACCurrentDensity "Metal1"
        (( "frequency" nil nil ))
        ( list(
             (1       5e-07)
             (2e+08   4e-07)
           )
        )
    )
) ;currentDensityTables
```

  If the index design value (`frequency`) is greater than or equal to `1`, the layer attribute value is `5e-07`; if the index design value is greater than or equal to `2e+08`, the attribute value is `4e-07`. The value returned is the first value in the table that matches a query of the lookup table.

Note: Current density tables do not support expressions for indexes; you cannot parameterize an index by specifying it as a techParams value (`techParam(`*`paramName`*`)`).

peakACCurrentDensity

```
layerRules(
     currentDensity(
          ( peakACCurrentDensity tx_layer f_value )
          …
     ) ;currentDensity

     currentDensityTables(
          ( peakACCurrentDensity tx_layer
               (( "frequency" nil nil ["width" | "cutArea" nil nil] )
                [f_default]
               )
               (g_table)
          )
          …
     ) ;currentDensityTables
) ;layerRules
```

Description

Sets the peak AC current density for the specified layer, absolutely or conditionally, as a function of the frequency (1-D table) or frequency and width of the object (2-D table) on the layer. The value specified by this rule is the maximum (peak) AC current that an object on the layer can handle.

Arguments

tx_layer	The layer on which the rule is applied. Valid values: The layer name or the layer number
f_value	The maximum AC current density, in milliamps per square micron.
f_default	The current density value to be used when no table entry applies.
g_table	The format of a row in the current density table is as follows:

```
( f_frequency f_width | f_cutArea f_value )
```

where,

- *f_frequency* is the frequency, in megahertz.
- *f_width* is the width of a metal layer.
- *f_cutArea* is the cut area for a cut layer.
- *f_value* is the maximum AC current density, in milliamps per square micron.

Related Topics

- Current Density and Current Density Tables
- layerRules

avgACCurrentDensity

```
layerRules(
     currentDensity(
          ( avgACCurrentDensity tx_layer f_value )
          …
     ) ;currentDensity

     currentDensityTables(
          ( avgACCurrentDensity tx_layer
               (( "frequency" nil nil ["width" | "cutArea" nil nil] )
                [f_default]
               )
               (g_table)
          )
          …
     ) ;currentDensityTables
) ;layerRules
```

Description

Sets the average AC current density for the specified layer, absolutely or conditionally, as a function of the frequency (1-D table) or frequency and width of the object (2-D table) on the layer. The value specified by this rule is the average AC current that an object on the layer can handle.

Arguments

`tx_layer`	The layer on which the rule is applied. Valid values: The layer name or the layer number
`f_value`	The average AC current density, in milliamps per square micron.
`f_default`	The current density value to use when no table entry applies.
`g_table`	The format of a row in the current density table is as follows: `( f_frequency f_width | f_cutArea f_value )` where, ■ *`f_frequency`* is the frequency, in megahertz. ■ *`f_width`* is the width of a metal layer. ■ *`f_cutArea`* is the cut area for a cut layer. ■ *`f_value`* is the average AC current density, in milliamps per square micron.

Related Topics

- Current Density and Current Density Tables
- layerRules

rmsACCurrentDensity

```
layerRules(
     currentDensity(
          ( rmsACCurrentDensity tx_layer f_value )
          …
     ) ;currentDensity

     currentDensityTables(
          ( rmsACCurrentDensity tx_layer
               (( "frequency" nil nil ["width" | "cutArea" nil nil] )
                [f_default]
               )
               (g_table)
          …
          )
     ) ;currentDensityTables
) ;layerRules
```

Description

Sets the root mean square (RMS) AC current density for the specified layer, absolutely or conditionally, as a function of the frequency (1-D table) or frequency and width of the object (2-D table) on the layer. The value specified by this rule is the maximum RMS AC current that an object on the layer can handle.

Arguments

`tx_layer`	The layer on which the rule is applied. Valid values: The layer name or the layer number
`f_value`	The maximum RMS AC current density, in milliamps per square micron.
`f_default`	The current density value to use when no table entry applies.
`g_table`	The format of a row in the current density table is as follows:

```
( f_frequency f_width | f_cutArea f_value )
```

where,

- *`f_frequency`* is the frequency, in megahertz.
- *`f_width`* is the width of a metal layer.
- *`f_cutArea`* is the cut area for a cut layer.
- *`f_value`* is the maximum RMS AC current density, in milliamps per square micron.

Related Topics

- Current Density and Current Density Tables
- layerRules

avgDCCurrentDensity

```
layerRules(
     currentDensity(
          ( avgDCCurrentDensity tx_layer f_value )
          …
          ) ;currentDensity

     currentDensityTables(
          ( avgDCCurrentDensity tx_metalLayer
               (( "width" nil nil )
                [f_default]
               )
               (g_table)
          )
          ( avgDCCurrentDensity tx_cutLayer
               (( "cutArea" nil nil )
                [f_default]
               )
               (g_table)
          )
     ) ;currentDensityTables
) ;layerRules
```

Description

Sets the average DC current density for the specified layer, absolutely or conditionally, as a function of the width or the area of the object. The value specified by this rule is the average DC current that an object on the layer can handle.

Arguments

`tx_layer`	The layer on which the rule is applied. Valid values: The layer name or the layer number
`f_value`	The maximum average DC current density, in milliamps per square micron.
`tx_metallayer`	The metal layer on which to apply the rule. Valid values: The layer name or the layer number
`tx_cutlayer`	The cut layer on which to apply the rule. Valid values: The layer name or the layer number
`f_default`	The current density value to use when no table entry applies.
`g_table`	The format of a row in the current density table is as follows: `(` *`f_width f_value`* `)` or `(` *`f_cutArea f_value`* `)` where, ■ *`f_width`* is the width. ■ *`f_cutArea`* is the area. ■ *`f_value`* is the average DC current density, in milliamps per square micron.

Related Topics

- Current Density and Current Density Tables
- layerRules

stampLabelLayers

```
layerRules(
     stampLabelLayers(
          ( textLayer layers)
          …
     ) ;stampLabelLayers
) ;layerRules
```

Description

Defines a rule for creating a label on a particular layer-purpose pair. It specifies text (label) layers and the corresponding layer-purpose pair (LPP) entries of connected type associated with them.

The text name is considered as the name of the electrical node when the text origin is on or inside a shape on one of the listed layers. A label with an origin that is positioned over multiple conducting layers is applied to the first layer or LPP in the layer list; that is, the first layer or LPP specified in the list takes the highest precedence and the last one takes the lowest precedence.

Arguments

textLayer	The layer associated with the subsequently specified layers of connected type. Valid values: The layer name or the layer number
layers	A layer or LPP of connected type.

Example

```
stampLabelLayers(
    ( Metal1 (Metal1 fill) )
    ((Metal2 pin) Metal2)
    ( Text (Metal2 pin) Metal2 (Metal1 pin) Metal1)
) ;stampLabelLayers
```

The first layer-purpose pair entry of a rule indicates the layer-purpose pair on which the label is created for a shape that exists on layer-purpose pairs defined by the remaining entries of the rule.

Note: If a purpose is not specified in the first entry, the valid list of purposes for the specified layer is considered for creating the label.

Related Topics

- layerRules

labelLayers

```
layerRules(
     labelLayers(
          ( tx_lableLayer1 tx_conLayer1 [ tx_conLayer2 … ] )
          ( tx_labelLayer2 tx_conLayer … )
          …
     ) ;labelLayers
) ;layerRules
```

Description

Specifies how text on label layers and the layers of any type associated with them should be handled.

The text name is not considered for naming electrical nodes. The label is associated with the first layer that has a shape that touches or overlaps the text. This specification provides a systematic means for establishing parent (shape)/child (text) relationships that are more general than a standard database parent/child relationship; with the `labels` relationship, the parent could be lower in the hierarchy and can be one of many layers.

Arguments

tx_labelLayer	The layer associated with the subsequently specified layers. Valid values: The layer name or the layer number
tx_conLayer	A layer of any type to be associated with the label layer.

Example

```
labelLayers(
    ( text Metal17 Metal16 Metal15 Poly Oxide )
) ;labelLayers
```

Related Topics

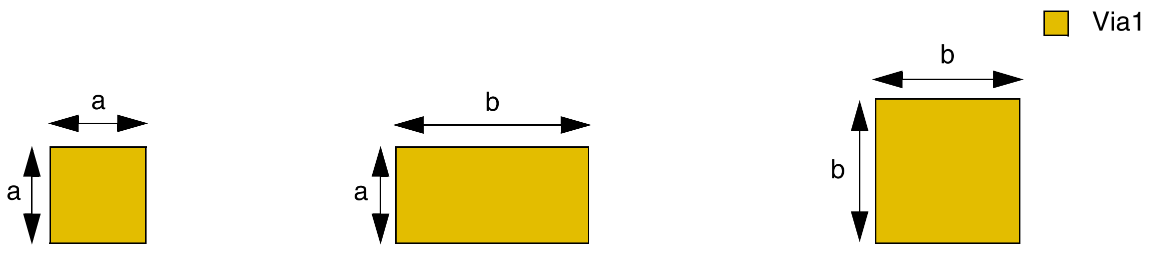
- layerRules

cutClasses

```
layerRules(
     cutClasses( [t_name]
          ( tx_cutLayer
               ( t_cutClassName
                    ['numCuts x_numCuts]
                    ['minWidth]
                    ['minLength]
                    ['fixedOrientation]
                    ['cutClassGroups l_groups]
                    { f_width | (f_width f_length) }
               ) ;t_cutClassName
               …
          ) ;tx_cutLayer
          …
          )
          …
     ) ;cutClasses
) ;layerRules
```

Defines a cut class by specifying the exact width and length the members of that cut class can have on a layer. Each cut class on a layer is assigned a unique name by which it can be referenced. If *`length`* for a cut class is not specified, cut shapes belonging to that cut class are squares with both dimensions equal to the specified *`width`*.

The value *`numCuts`* defines the equivalent number of via cuts required to satisfy the `minNumCut` constraint and to calculate the resistance. If the equivalent number of via cuts is *`x`* and `minNumCut` requires *`n`* via cuts, `minNumCut` is satisfied if *`n`*/*`x`* rounded up to the nearest integer is greater than or equal to *`n`*. For example, if the equivalent number of cuts is `2` and `minNumCut` requires `2` via cuts, then `1` via cut would be sufficient to satisfy `minNumCut`.



Cut class `VA` with width 0.3

Cut class `VB` with width 0.3 and length 0.5

Cut class `VC` with width 0.5 and number of equivalent cuts equal to 4. Cut shape of class `VC` satisfies `minNumCut` = `3`.

Cut class constraints can be defined only in a technology database.

A technology file can include multiple cutClass sections, each within a uniquely named constraint group. A cutClass section can be used without a name to indicate that the cutClass definitions in it be used to resolve cutClass references in constraints in the foundry constraint group. If all cutClass sections are named, the cutClass definitions in the last cutClass section are used to resolve such references.

Naming cutClass sections can be useful when defining a set of cut classes to be used in an alternate foundry constraint group. The named cutClass constraint group can be added as a member constraint group in another constraint group, which in turn, can be used as an alternate foundry constraint group.

Cut class constraints are referenced by other constraints to restrict these other constraints to apply only to cut shapes of the specified size on the specified cut layer. For example, in some processes, different allowed spacings are required for via cuts on a layer depending on the sizes of the via cuts.

All cut class constraints for cut layers defined in a single technology database must be added in one OR constraint group, which, in turn, must be a member of the `foundry` constraint group in the same technology database. The technology file loader creates a constraint for each cut class defined in the technology database and adds all of these constraints to a constraint group with the name `cutClass_<`*`libName`*`>`, which is the name used when *`t_name`* is not specified. If cut classes are defined in different technology databases contained in an ITDB, all cut classes are considered to be part of the same effective OR constraint group.

If multiple cut classes with the same dimensions exist in the technology database, the cut class names in the `minCutClassSpacing (One layer)` and `minCutClassSpacing (Two layers)` constraints in the dumped technology file are not the same as in the input technology file. The technology dumper randomly picks any one name.

Values

`f_width` `|` `(`*`f_width f_length`*`)`	The width and length of cut shapes belonging to the cut class must be equal to these values.

Parameters

`t_name`	The name of the cut class constraint group.

tx_cutLayer	The cut layer on which the constraint is applied. You can specify more than one cut layers, each enclosed in double quotes and separated by a space. Type: String (layer name) or Integer (layer number)
t_cutClassName	The cut class name that must be unique for a given layer.
`'numCuts` *g_numCuts*	The number of equivalent cuts for the cut class.
`'minwidth`	The width of the cut shape must be greater than or equal to *width*. Otherwise, the width of the cut shape must be equal to *width*.
`'minLength`	The length of the cut shape must be greater than or equal to *length*. Otherwise, the length of the cut shape must be equal to *length*.
`'fixedOrientation`	(ICADVM20.1 Only – 95512) Of the two constraint values, the first is always interpreted as the x-span of the cut class and the second is always interpreted as the y-span of the cut class.
`'cutClassGroups` *l_groups*	The cut class belongs to the cut class groups in this list. The groups are listed by name. Applications that look at via cuts across layers would pick one member on each layer represented in the group; applications that look at only one layer, would ignore group members on other layers.

Examples

- Example 1: cutClasses with numCuts, minWidth, and minLength
- Example 2: cutClasses with cutClassGroups

Example 1: cutClasses with numCuts, minWidth, and minLength

You can define all cut classes for a given layer together, as shown below. Four cut classes, namely, `VA`, `VB`, `VC`, and `VD`, are created in this example.

```
layerRules(
    cutClasses(
        ;( layerName )
        ;( (cutClassName (width length)) )
        ;( --------------------------------------------------------------- )
        ( Via1
            ( VA (0.3 0.4) )
            ( VB 'numCuts 2 (0.4 0.6) )
            ( VC 'minWidth 'minLength (0.8 0.7) )
            ( VD (0.3 0.3) )
        )
        ( Via3
            ( VA (0.4 0.3) )
            ( VB 'numCuts 2 (0.5 0.6) )
            ( VC 'minWidth 'minLength (0.8 0.7) )
            ( VD (0.4 0.4) )
        )
    ) ;cutClasses
) ;layerRules
```

Example 2: cutClasses with cutClassGroups

You can define constraint groups as shown below. Two cut class groups, named `g1` and `g2`, are defined in this example.

A `viaStack` application would pick one member on each layer. For `g1`, it can pick `V0A`, `V1A`, `V2A` or `V0A`, `V1B`, `V2A`. Whereas, for `g2`, it would pick `V0A`, `V1C`, `V2B`. If a constraint is specified on layer `Via1`, group `g1` would refer to `V1A` and `V1B`.

```
cutClasses(
        ( V0
            (V0A 'numCuts 1 'cutClassGroups ("g1" "g2") (0.034 0.034))
        )
        ( Via1
            (V1A 'numCuts 1 'cutClassGroups ("g1") (0.034 0.034))
            (V1B 'numCuts 2 'cutClassGroups ("g1") (0.07  0.034))
            (V1C 'numCuts 2 'cutClassGroups ("g2") (0.07  0.034))
        )
        ( Via2
            (V2A 'numCuts 1 'cutClassGroups ("g1") (0.05 0.05))
            (V2B 'numCuts 2 'cutClassGroups ("g2") (0.1  0.05))
        )
) ;cutClasses
```

Note: For more information, see Create Via Options Form in *Virtuoso® Layout Suite L User Guide*.

Accessing Cut Class Information Using SKILL

You can access information about cut classes from the technology file using SKILL. To do this, you need to perform the following steps:

1. Obtain `techID` from an open cellview; for example:

   ```
   cv = geGetEditCellView()
   tf = techGetTechFile(cv)
   ```

   Or, obtain `techID` by specifying the technology library name; for example:

   ```
   tf=techGetTechFile(ddGetObj("gpdk045"))
   ```

2. Use `cst` to obtain the `foundry` constraint group; for example:

   ```
   foundry_cg = cstFindConstraintGroupIn(tf "foundry")
   ```

   The sub-constraint group that contains the cut classes is the first element of the `foundry` constraint group.

   ```
   cutClass_cg = car( setof( cg foundry_cg~>objects cg~>defName == "cutClass"))
   ```

3. In CIW, type `cutClass_cg~>??`, which returns:

   ```
   (cst:0xf082b294 tech db:0xf082b612 cellView nil
       objType "constraintGroup" name "cutClass_gpdk090" defName
       "cutClass" operator or owner nil
   ```

```
        objects
        (cst:0xf082a492 cst:0xf082a493 cst:0xf082a494 cst:0xf082a495
         cst:0xf082a496 cst:0xf082a497 cst:0xf082a498 cst:0xf082a499
         cst:0xf082a49a cst:0xf082a49b cst:0xf082a49c cst:0xf082a49d
         cst:0xf082a49e
        )
    )
```

4. In CIW, type `cutClass_cg ~>objects~>??`, which returns all the cut classes defined in the technology library; for example:

```
(cst:0xf082a492 tech db:0xf082b612 cellView nil
    objType "layerConstraint" name "C__0" defName
    "cutClass" layers
    ("Via1") value
    (0.14 0.14)
    params
    (className("Small")
        (numCuts 1)
    ) hard t ID
    nil description nil
)
(cst:0xf082a493 tech db:0xf082b612 cellView nil
    objType "layerConstraint" name "C__1" defName
    "cutClass" layers
    ("Via1") value
    (0.14 0.28)
    params
    (className("Bar")
        (numCuts 2)
    ) hard t ID
    nil description nil
)
(cst:0xf082a494 tech db:0xf082b612 cellView nil
    objType "layerConstraint" name "C__2" defName
    "cutClass" layers
    ("Via1") value
    (0.14 0.28)
    params
    (className("Bar1")
        (numCuts 2)
    ) hard t ID
    nil description nil
)
```

5. Run `cutClass_cg~>objects`, to obtain a list of cut class IDs; for example:

```
(cst:0xf082a492 cst:0xf082a493 cst:0xf082a494 cst:0xf082a495 cst:0xf082a496
 cst:0xf082a497 cst:0xf082a498 cst:0xf082a499 cst:0xf082a49a cst:0xf082a49b
 cst:0xf082a49c cst:0xf082a49d cst:0xf082a49e
)
```

Related Topics

layerRules

6

Technology File Site Definitions

This chapter contains the following topics pertaining to site definitions:

- siteDefs
- scalarSiteDefs
- arraySiteDefs

siteDefs

```
siteDefs()
```

Description

Specifies site definitions. Subsections specifying site definitions must be enclosed within the parentheses of this section.

scalarSiteDefs

```
scalarSiteDefs(
     ( t_siteDefName
       t_siteDefType
       n_width
       n_height
       [g_symmetricInX]
       [g_symmetricInY]
       [g_symmetricInR90]
     )
     …
) ;scalarSiteDefs
```

Description

Specifies scalar site definitions, each of which defines a site where you can place cells in a row.

Arguments

t_siteDefName	The site definition name. Valid values: Any string
t_siteDefType	The site definition type. Valid values: `pad`, `core`
n_width	The width of the site. Valid values: Any number
n_height	The height of the site. Valid values: Any number
g_isSymmetricInX	Specifies whether the scalar site definition is symmetric in the X direction. Valid values: `t`, `nil` Default: `nil`
g_isSymmetricInY	Specifies whether the scalar site definition is symmetric in the Y direction. Valid values: `t`, `nil` Default: `nil`

`g_isSymmetricInR90`	Specifies whether the scalar site definition is symmetric in rotation. Valid values: `t`, `nil` Default: `nil`

Example

```
siteDefs(
    scalarSiteDefs(
        (coreSite  core  574.84   1352.96   t   nil   t )
        (padSite   pad   1783.84  845.84   nil   t   nil)
    ) ;scalarSiteDefs
) ;siteDefs
```

Related Topics

- siteDefs

arraySiteDefs

```
arraySiteDefs(
     ( t_siteDefName
       t_siteDefType
       l_sitePattern
       [g_symmetricInX]
       [g_symmetricInY]
       [g_symmetricInR90]
     )
     …
) ;arraySiteDefs
```

Description

Defines an array of scalar site definitions.

Arguments

t_siteDefName	The array site definition name. Valid values: Any string
t_siteDefType	The type of scalar site definitions to include in the array. Valid values: `pad`, `core` **Note:** All scalar site definitions in an array must be of the same type.

`l_sitePattern`	A list specifying each scalar site definition in the array, along with its offset from the origin (the lower-left corner) of the array and its orientation. This argument has the following syntax:

```
((t_scalarSiteDefName g_xOffset g_yOffset
t_orient)…)
```

where,

- *`t_scalarSiteDefName`* is the name of the scalar site definition included in the array.
- *`g_xOffset`* is the X offset for the scalar site definition.
- *`g_yOffset`* is the Y offset for the scalar site definition.
- *`t_orient`* is the orientation of the scalar site definition.
  Valid values: `R0`, `R90`, `R180`, `R270`, `MX`, `MI`, `MXR90`, `MYR90`

  where,

`R0`	Maintain scalar site orientation
`R90`	Rotate 90 degrees
`R180`	Rotate 180 degrees
`R270`	Rotate 270 degrees
`MX`	Mirror about the x axis
`MI`	Mirror about the y axis
`MXR90`	Mirror about the x axis and then rotate 90 degrees
`MYR90`	Mirror about the y axis and then rotate 90 degrees

`g_isSymmetricInX`	Specifies whether the array site definition is symmetric in the X direction. Valid values: `t`, `nil` Default: `nil`
`g_isSymmetricInY`	Specifies whether the array site definition is symmetric in the Y direction. Valid values: `t`, `nil` Default: `nil`
`g_isSymmetricInR90`	Specifies whether the array site definition is symmetric in rotation. Valid values: `t`, `nil`

Example

```
siteDefs(
    arraySiteDefs(
        (myCoreArray  core
          (
            (coreSite  0.0  0.0  R270)
            (coreSite  575  0.0  MY)
          )
          t nil t
        )

        (myPadArray  pad
          (
            (padSite  0.0   0.0  MYR90)
            (padSite  1784  0.0  R180)
          )
          nil t nil
        )
    ) ;arraySiteDefs
) ;siteDefs
```

Related Topics

- siteDefs

7

Technology File Via Definitions and Via Specifications

This chapter contains the following topics pertaining to via definitions and via specifications:

viaDefs

```
viaDefs()
```

Description

Contains via definitions and via variants. All subsections specifying via definitions and via variants must be enclosed within the parentheses of this section.

standardViaDefs

```
standardViaDefs(
     ( t_viaDefName tx_layer1 tx_layer2
          ( tx_cutLayer n_cutWidth n_cutHeight [n_resistancePerCut] )
          ( x_cutRows x_cutCols ( l_cutSpace ) [(l_cutPattern)])
          (l_layer1Enc) (l_layer2Enc)
          ( l_layer1Offset ) ( l_layer2Offset ) ( l_origOffset )
          [tx_implant1 ( l_implant1Enc )
               [tx_implant2 ( l_implant2Enc ) [ tx_wellSubstrate ] ]
          ]
     )
) ;standardViaDefs
```

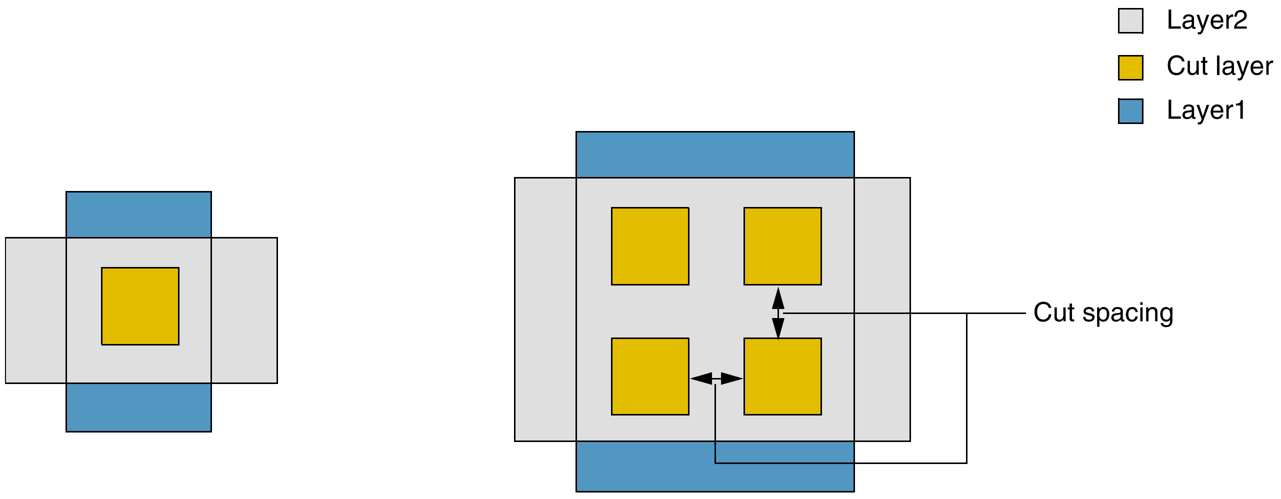
Description

Specifies standard via definitions. A standard via definition has a fixed set of parameters.

An advantage of using a standard via definition is that it is fully captured in the technology file, which makes it easier to manage the via definition.

Important

`standardViaDefs` are more efficient than customViaDefs because they do not need to access a separate via master design and an external evaluation to determine geometries.



Arguments

`t_viaDefName`	The standard via definition name. Valid values: Any string; it must be unique in the technology database graph
`tx_layer1`	Preferably, but not necessarily, the bottom routing layer of the via. Valid values: The layer name or the layer number
`tx_layer2`	Preferably, but not necessarily, the top routing layer of the via. Valid values: The layer name or the layer number
`tx_cutLayer`	The cut layer. Valid values: The layer name or the layer number; the layer function must be `cut` **Note:** The mask number of the bottom layer must be lower than the mask number of the cut layer, which, in turn, must be lower than the mask number of the top layer.
`f_cutWidth`	The width of a cut. Valid values: Any floating-point number
`f_cutHeight`	The height of a cut. Valid values: A floating-point number
`n_resistancePerCut`	The resistance per cut. Valid values: Any number
`x_cutRows`	The number of cut rows. Valid values: An integer
`x_cutCols`	The number of cut columns. Valid values: An integer
`l_cutSpace`	A list defining the distance between cuts. The syntax is as follows: `(` *`f_width f_height`* `)` where, *`f_width`* is the spacing between cuts in the X direction, and *`f_height`* is the spacing between cuts in the Y direction. Valid values: A floating-point number
`l_cutPattern`	A list representing the cuts in each row. Valid values: Either `0` or `1`

l_layer1Enc	A list defining the distance from the edge of the cut layer to the edge of *layer1*. The syntax is as follows: `( f_width f_height )` where, *f_width* is the distance from the cut edge to the edge of *layer1* in the X direction, and *f_height* is the distance from the cut edge to the edge of *layer1* in the Y direction. Valid values: A floating-point number
l_layer2Enc	A list defining the distance from the edge of the cut layer to the edge of *layer2*. The syntax is as follows: `( f_width f_height )` where, *f_width* is the distance from the cut edge to the edge of *layer2* in the X direction, and *f_height* is the distance from the cut edge to the edge of *layer2* in the Y direction. Valid values: A floating-point number
l_layer1Offset	A list defining the offset for *layer1*. The syntax is as follows: `( f_width f_height )` where, *f_width* is the horizontal offset, and *f_height* is the vertical offset. Valid values: A floating-point number
l_layer2Offset	A list defining the offset for *layer2*. The syntax is as follows: `( f_width f_height )` where, *f_width* is the horizontal offset, and *f_height* is the vertical offset. Valid values: A floating-point number
l_origOffset	A list defining the offset for the origin of the via. The syntax is as follows: `( f_width f_height )` where, *f_width* is the horizontal offset, and *f_height* is the vertical offset. Valid values: A floating-point number
tx_implant1	The first implant layer of the via. Valid values: The layer name or the layer number
l_implant1Enc	A list defining the distance from the edge of *layer1* to the edge of *implant1* layer. The syntax is as follows: `( f_width f_height )` where, *f_width* is the distance from the edge of *layer1* to the edge of *implant1* layer in the X direction, and *f_height* is the distance from the edge of *layer1* to the edge of *implant1* layer in the Y direction. Valid values: A floating-point number

`tx_implant2`	The second implant layer of the via. Valid values: The layer name or the layer number
`l_implant2Enc`	A list defining the distance from the edge of *`layer2`* to the edge of *`implant2`* layer. The syntax is as follows: `(` *`f_width f_height`* `)` where, *`f_width`* is the distance from the edge of *`layer2`* to the edge of *`implant2`* layer in the X direction, and *`f_height`* is the distance from the edge of *`layer2`* to the edge of *`implant2`* layer in the Y direction. Valid values: A floating-point number
`tx_wellSubstrate`	When specified, placing the via on the specified well causes the via to be an electrical connection to the well. Valid values: A well that is of type `nwell` or `pwell` or of name `substrate`

Example

```
standardViaDefs(
; ( viaDefName layer1 layer2 ( cutLayer cutWidth cutHeight [ resistancePerCut ] )
; ( cutRows cutCol ( cutSpace ) [( l_cutPattern )])
; ( layer1Enc )  (layer2Enc)  ( layer1Offset ) ( layer2Offset ) ( origOffset )
; [implant1 (implant1Enc) [implant2 (implant2Enc) [wellSubstrate]]])

  ( M2xM1xG   M1   M2    ( "VIA1"  0.13  0.13  5.0 )
   ( 1   1   ( 0.15  0.15 ) )
   ( 0.05  0.005 )  ( 0.05  0.005 )  ( 0.0  0.0 )  ( 0.0  0.0 )  ( 0.0  0.0 )
   pimplant  (1.0  1.0)
  )


  ( M1_P   diff   metal1   ("cont"  0.6   0.6 )
   ( 2   4   ( 0.6  0.6 )  (( 1 0  1 0 )  ( 1 1   0 1 )))
   ( 0.6  0.6 )  ( 0.6  0.6 )  ( 0.0   0.0 )  ( 0.0  0.0 )  ( 0.0  0.0 )

  )


  ( M3xM2xG   M2   M3   ( "VIA2"  0.13  0.13  5.0 )
   ( 1   1  ( 0.15 0.15 ) )
   ( 0.05  0.005 )  ( 0.05  0.005 )  ( 0.0  0.0 )  ( 0.0  0.0 )  ( 0.0  0.0 )
  )


  ( M4xM3xG   M3   M4   ( "VIA3"  0.13  0.13  5.0 )
   ( 1   1   ( 0.15  0.15 ) )
   ( 0.05  0.005 )  ( 0.05  0.005 )  ( 0.0  0.0 )  ( 0.0  0.0 )  ( 0.0  0.0 )
  )
  ( M5xM4xG   M4   M5   ( "VIA4"  0.13  0.13  5.0 )
   ( 1   1   ( 0.15  0.15 ) )
   ( 0.05  0.005 )  ( 0.05  0.005 )  ( 0.0  0.0 )  ( 0.0  0.0 )  ( 0.0  0.0 )
  )

  ( M6xM5xG   M5   M6   ( "VIA5"  0.36  0.36  5.0 )
   ( 1   1   ( 0.29  0.29 ) )
   ( 0.09  0.03 )  ( 0.09  0.03 )  ( 0.0  0.0 )  ( 0.0  0.0 )  ( 0.0  0.0 )
  )

  ( PTAP   pdiff   M1   ( "contact"  1.0  1.0 )
   ( 1   1   ( 1.0  1.0 ) )
   ( 1.0  1.0 )  ( 0.5  0.5 )  ( 0.0  0.0 )  ( 0.0  0.0 )  ( 0.0  0.0 )
     nil  nil  nil  nil  substrate
  )

  ( NTAP   ndiff   M1   ( "contact"  1.0  1.0 )
   ( 1   1   ( 1.0  1.0 ) )
   ( 1.0  1.0 )  ( 0.5  0.5 )  ( 0.0  0.0 )  ( 0.0  0.0 )  ( 0.0  0.0 )
     nil  nil  nil  nil  nwell
  )
) ;standardViaDefs
```

customViaDefs

```
customViaDefs(
     ( t_viaDefName
       t_libName
       t_cellName
       t_viewName
       tx_layer1
       tx_layer2
       n_resistancePerCut
     )
) ;customViaDefs
```

Description

Specifies custom via definitions. To specify a custom via definition, the cellview on which it is based must exist in the specified library. Unlike `standardViaDefs` that describe all the via layers and dimensions, `customViaDefs` define only a list of existing cellviews.

Custom via definitions do not define geometric parameters; they define only the name of the via and the <lib | cell | view> information.

Note: The technology dumper outputs `customViaDef` with a leading ";" comment when the following criteria is met:

- It is referenced by `tcDeclareDevice` or any other device creation construct in the technology file.
- The `tcDeclareDevice` definition references a `customViaDef` in the technology database library.

Arguments

t_viaDefName	The custom via definition name. Valid values: Any string; it must be unique in the technology database graph
t_libName	The name of the design library that contains the cellview of the via device. Valid values: Any valid design library name
t_cellName	The name of the cell for the via definition. Valid values: Any valid cell name
t_viewName	The name of the view of the cell for the via definition. Valid values: Any valid view name
tx_layer1	The bottom routing layer of the via. Valid values: The layer name or the layer number
tx_layer2	The top routing layer of the via. Valid values: The layer name or the layer number **Note:** The mask number of the bottom layer must be lower than the mask number of the top layer.
n_resistancePerCut	The resistance per cut. Valid values: Any number Default: `0.0`

Example

```
customViaDefs(
; ( viaDefName        libName     cellName     viewName  layer1 layer2
;   resistancePerCut )
  ( M2_M1_H           testTech4   M2_M1_H           via      M1   M2   5.0 )
  ( M2_M1_H           "2testTech" M2_M1_H           via      M1   M2   5.0 )
  ( M2_M1_V1          testTech4   M2_M1_V1          via      M1   M2   5.0 )
  ( M2_M1_V2          testTech4   M2_M1_V2          via      M1   M2   5.0 )
  ( M2_M1_DBLCUT_E    testTech4   M2_M1_DBLCUT_E    via      M1   M2   5.0 )
  ( M2_M1_DBLCUT_W    testTech4   M2_M1_DBLCUT_W    via      M1   M2   5.0 )
  ( M2_M1_QUADCUT_NE  testTech4   M2_M1_QUADCUT_NE  via      M1   M2   5.0 )
  ( M2_M1_QUADCUT_NW  testTech4   M2_M1_QUADCUT_NW  via      M1   M2   5.0 )
  ( M2_M1_QUADCUT_SE  testTech4   M2_M1_QUADCUT_SE  via      M1   M2   5.0 )
  ( M2_M1_QUADCUT_SW  testTech4   M2_M1_QUADCUT_SW  via      M1   M2   5.0 )
  ( M3_M2_SINGLE      testTech4   M3_M2_SINGLE      via      M2   M3   5.0 )
  ( M3_M2_V1          testTech4   M3_M2_V1          via      M2   M3   5.0 )
  ( M3_M2_V2          testTech4   M3_M2_V2          via      M2   M3   5.0 )
  ( M3_M2_TOS         testTech4   M3_M2_TOS         via      M2   M3   5.0 )
  ( M3_M2_DBLCUT_E    testTech4   M3_M2_DBLCUT_E    via      M2   M3   5.0 )
  ( M3_M2_DBLCUT_W    testTech4   M3_M2_DBLCUT_W    via      M2   M3   5.0 )
  ( M3_M2_QUADCUT_NE  testTech4   M3_M2_QUADCUT_NE  via       M2    M3    5.0 )
  ( M3_M2_QUADCUT_NW  testTech4   M3_M2_QUADCUT_NW  via       M2    M3    5.0 )
  ( M3_M2_QUADCUT_SE  testTech4   M3_M2_QUADCUT_SE  via       M2    M3    5.0 )
  ( M3_M2_QUADCUT_SW  testTech4   M3_M2_QUADCUT_SW  via       M2    M3    5.0 )
  ( M4_M3_SINGLE      testTech4   M4_M3_SINGLE      via       M3    M4    5.0 )
  ( M4_M3_V1          testTech4   M4_M3_V1          via       M3    M4    5.0 )
  ( M4_M3_V2          testTech4   M4_M3_V2          via       M3    M4    5.0 )
  ( M4_M3_TOS         testTech4   M4_M3_TOS         via       M3    M4    5.0 )
  ( M4_M3_DBLCUT_N    testTech4   M4_M3_DBLCUT_N    via       M3    M4    5.0 )
  ( M4_M3_DBLCUT_S    testTech4   M4_M3_DBLCUT_S    via       M3    M4    5.0 )
  ( M4_M3_QUADCUT_NE  testTech4   M4_M3_QUADCUT_NE  via       M3    M4    5.0 )
  ( M4_M3_QUADCUT_NW  testTech4   M4_M3_QUADCUT_NW  via       M3    M4    5.0 )
  ( M4_M3_QUADCUT_SE  testTech4   M4_M3_QUADCUT_SE  via       M3    M4    5.0 )
  ( M4_M3_QUADCUT_SW  testTech4   M4_M3_QUADCUT_SW  via       M3    M4    5.0 )
  ( M5_M4_SINGLE      testTech4   M5_M4_SINGLE      via       M4    M5    5.0 )
  ( M5_M4_V1          testTech4   M5_M4_V1          via       M4    M5    5.0 )
  ( M5_M4_V2          testTech4   M5_M4_V2          via       M4    M5    5.0 )
  ( M5_M4_TOS         testTech4   M5_M4_TOS         via       M4    M5    5.0 )
  ( M5_M4_DBLCUT_E    testTech4   M5_M4_DBLCUT_E    via       M4    M5    5.0 )
  ( M5_M4_DBLCUT_W    testTech4   M5_M4_DBLCUT_W    via       M4    M5    5.0 )
  ( M5_M4_QUADCUT_NE  testTech4   M5_M4_QUADCUT_NE  via       M4    M5    5.0 )
  ( M5_M4_QUADCUT_NW  testTech4   M5_M4_QUADCUT_NW  via       M4    M5    5.0 )
  ( M5_M4_QUADCUT_SE  testTech4   M5_M4_QUADCUT_SE  via       M4    M5    5.0 )
  ( M5_M4_QUADCUT_SW  testTech4   M5_M4_QUADCUT_SW  via       M4    M5    5.0 )
  ( M6_M5_SINGLE      testTech4   M6_M5_SINGLE      via       M5    M6    5.0 )
  ( M6_M5_DBLCUT_N    testTech4   M6_M5_DBLCUT_N    via       M5    M6    5.0 )
  ( M6_M5_DBLCUT_S    testTech4   M6_M5_DBLCUT_S    via       M5    M6    5.0 )
  ( M6_M5_DBLCUT_E    testTech4   M6_M5_DBLCUT_E    via       M5    M6    5.0 )
  ( M6_M5_DBLCUT_W    testTech4   M6_M5_DBLCUT_W    via       M5    M6    5.0 )
) ;customViaDefs
```

Important

A name that starts with a number should be enclosed in double quotes for the technology file compiler to compile it successfully. For example, if your technology file contains a custom via definition such as `(V12_HV_via 0techlib V12_HV via METAL1 METAL2 4.0)`, you must enclose the library name starting with a number, `0techlib`, in double quotes; for example, `"0techlib"`.

Similarly, any string type entry, such as `0Metal`, that starts with a number, must also be defined as `"0Metal"`.

standardViaVariants

```
standardViaVariants(
     ( t_viaVariantName t_viaDefName (tx_cutLayer f_cutWidth f_cutHeight)
          (x_cutRows x_cutCol (l_cutSpace))
          (l_layer1Enc) (l_layer2Enc) (l_layer1Offset) (l_layer2Offset)
          (l_origOffset)
          (l_implant1Enc) (l_implant2Enc) (l_cutPattern)
     )
) ;standardViaVariants
```

Description

Defines standard via variants. This subsection specifies the parameters applied to create a named variant of the specified standard via definition.

Important

The `standardViaVariants()` subsection must be placed after the `standardViaDefs()` subsection in the technology file.

Each variant must have a unique name and definition. Multiple variants of the same standard via must specify at least one unique parameter. This is because the technology file compiler rejects multiple identical variants of the same custom via.

Arguments

t_viaVariantName	The name of the standard via variant. Valid values: Any string; it must be unique in the technology database graph
t_viaDefName	The name of the standard via definition. Valid values: Any valid `standardViaDef` name
tx_cutLayer	The cut layer. Valid values: The layer name or the layer number; the layer function must be `cut` **Note:** The mask number of the bottom layer must be lower than the mask number of the cut layer, which, in turn, must be lower than the mask number of the top layer.
f_cutWidth	The width of a cut. Valid values: A floating-point number

`f_cutHeight`	The height of a cut. Valid values: A floating-point number
`x_cutRows`	The number of cut rows. Valid values: An integer
`x_cutCols`	The number of cut columns. Valid values: An integer
`l_cutSpace`	A list defining the distance between cuts. The syntax is as follows: `(` *`f_width f_height`* `)` where, *`f_width`* is the spacing between cuts in the X direction, and *`f_height`* is the spacing between cuts in the Y direction. Valid values: A floating-point number
`l_layer1Enc`	A list defining the distance from the edge of the cut layer to the edge of *`layer1`*. The syntax is as follows: `(` *`f_width f_height`* `)` where, *`f_width`* is the distance from the cut edge to the edge of *`layer1`* in the X direction, and *`f_height`* is the distance from the cut edge to the edge of *`layer1`* in the Y direction. Valid values: A floating-point number
`l_layer2Enc`	A list defining the distance from the edge of the cut layer to the edge of *`layer2`*. The syntax is as follows: `(` *`f_width f_height`* `)` where, *`f_width`* is the distance from the cut edge to the edge of *`layer2`* in the X direction, and *`f_height`* is the distance from the cut edge to the edge of *`layer2`* in the Y direction. Valid values: A floating-point number
`l_layer1Offset`	A list defining the offset for *`layer1`*. The syntax is as follows: `(` *`f_width f_height`* `)` where, *`f_width`* is the horizontal offset, and *`f_height`* is the vertical offset. Valid values: A floating-point number
`l_layer2Offset`	A list defining the offset for *`layer2`*. The syntax is as follows: `(` *`f_width f_height`* `)` where, *`f_width`* is the horizontal offset, and *`f_height`* is the vertical offset. Valid values: A floating-point number

l_origOffset	A list defining the offset for the origin of the via. The syntax is as follows: `( f_width f_height )` where, *f_width* is the horizontal offset, and *f_height* is the vertical offset. Valid values: A floating-point number
l_implant1Enc	A list defining the distance from the edge of *layer1* to the edge of *implant1* layer. The syntax is as follows: `( f_width f_height )` where, *f_width* is the distance from the edge of *layer1* to the edge of *implant1* layer in the X direction, and *f_height* is the distance from the edge of *layer1* to the edge of *implant1* layer in the Y direction. Valid values: A floating-point number
l_implant2Enc	A list defining the distance from the edge of *layer2* to the edge of *implant2* layer. The syntax is as follows: `( f_width f_height )` where, *f_width* is the distance from the edge of *layer2* to the edge of *implant2* layer in the X direction, and *f_height* is the distance from the edge of *layer2* to the edge of *implant2* layer in the Y direction. Valid values: A floating-point number
l_cutPattern	A list representing the cuts in each row. Valid values: Either `0` or `1`

Example

```
standardViaDefs(
; ( viaDefName layer1 layer2 ( cutLayer cutWidth cutHeight [ resistancePerCut ] )
; ( cutRows cutCol ( cutSpace   ) )
; ( layer1Enc )  (layer2Enc)    ( layer1Offset ) ( layer2Offset ) ( origOffset )
; [implant1 (implant1Enc) [implant2 (implant2Enc)]])

  ( M2xM1xG  M1  M2  ("VIA1"  0.13  0.13  5.0)
    (1  1  (0.15 0.15))
    (0.0 0.0) (0.0 0.0) (0.0 0.0)
    pimplant (1.0 1.0)
  )
) ;standardViaDefs
```

```
standardViaVariants(
; ( viaVariantName viaDefName ( cutLayer cutWidth cutHeight )
; ( cutRows cutCol ( cutSpace   ) )
; ( layer1Enc )  (layer2Enc)    ( layer1Offset ) ( layer2Offset ) ( origOffset )
; (implant1Enc) (implant2Enc) (cut_pattern]])

  ( M2xM1xG_Var1  M2xM1xG  ( "VIA1" 0.15 0.15 )
    (1  1  (0.2 0.2))
    (0.05 0.005) (0.05 0.005) (0.1 0.0) (0.0 0.0) (0.0 0.0)
    ( 1.0 1.0 ) nil ((1 1) (1 1))
  )
) ;standardViaVariants
```

Defines a variant named `M2xM1XG_Var1` of the standard via definition `M2xM1xG` with the following modified parameters:

- Sets the cut width and height for the cut layer `VIA1` to `0.15` (instead of 0.13)
- Sets the cut space for the cut rows and columns to `0.2` (instead of 0.15)
- Sets the *`layer1`* (`M1`) offset to (`0.1 0.0`) (instead of (`0.0 0.0`))

customViaVariants

```
customViaVariants(
     ( t_viaVariantName
       t_viaDefName
       l_viaParams
     )
) ;customViaVariants
```

Description

Defines custom via variants. The subsection specifies the parameters applied to create a named variant of the specified custom via definition.

Important

The `customViaVariants()` subsection must be placed after the `customViaDefs()` subsection in the technology file.

Each variant must have a unique name and definition and can take any list of parameter-value pairs. However, multiple variants of the same custom via must specify at least one unique parameter. This is because the technology file compiler rejects multiple identical variants of the same custom via.

Arguments

t_viaVariantName	The name of the custom via variant. Valid values: Any string; it must be unique in the technology database graph
t_viaDefName	The name of the custom via definition. Valid values: Any valid `customViaDef` name
l_viaParams	A list or lists of parameters to apply to the custom via variant. The syntax is as follows: `(` *`t_paramName g_paramValue`* `)` … where, *`paramName`* is the name of the parameter and *`paramValue`* is the value of the parameter **Note:** The technology file compiler does not perform checks on the parameters specified.

Examples

```
customViaDefs(
; ( viaDefName  libName    cellName  viewName  layer1  layer2  resistancePerCut )
  ( M2_M1_H     testTech4  M2_M1_H   via       M1      M2      5.0 )
  ( M2_M1_V1    testTech4  M2_M1_V1  via       M1      M2      5.0 )
) ;customViaDefs


customViaVariants(
; ( viaVariantName viaDefName  parameters )
  ( M2_M1_H_VAR1   M2_M1_H     (param1 0.0 )  (param2 1.0))
) ;customViaVariants
```

Defines a variant of the custom via definition `M2_M1_H` and adds parameters `param1` with value `0.0` and `param2` with value `1.0`.

```
;********************************
; DEVICES
;********************************
devices(
  tcCreateCDSDeviceClass()
  cdsViaDevice(
    ; (name cutLayer cutPurpose layer1 purpose1 layer2 purpose2
    ; row column origin stackedVias cutLayerW cutLayerL xCutSpacing yCutSpacing
    ; layer1XDirOverride layer1YDirOverride layer2XDirOverride layer2YDirOverride
    ; layer1Dir layer2Dir
    ; layer1XDefOverride layer1YDefOverride layer2XDefOverride layer2YDefOverride
    ; implantLayer1 implantLayer2 diffSpacing abutClass)
    ( viaM2V2M3 V2 drawing M2 drawing M3 drawing
        1 1 centerCenter t _def_ _def_ 0.05 0.05
        0.02 0.02 _def_ 0.008
        ; "Y" "X"
        "" ""
        0 0 0 0
        ; (
        ;   (("Nwell" "drawing") 1)
        ; )
        ; (
        ;   (("Nimp" "drawing") 0.5)
        ; )
        nil nil _def_ ""
        )
) ;devices


customViaDefs(
    ;( viaDefName libName cellName viewName layer1 layer2 resistancePerCut)
    ;( ---------- ------- -------- -------- ------ ------ ----------------)
     ( viaM2V2M3 techlib viaM2V2M3 layout M2 M3 0.0 )
) ;customViaDefs
```

```
customViaVariants(
    ;(viaVariantName viaDefName (paramName paramValue) …)
    ;( ------------------------------------------------)
     ( viaM2V2M3_LY1x1 viaM2V2M3
        ("row"    1)
        ("column" 1)
        ("xCutSpacing" 0.08)
        ("yCutSpacing" 0.08)
        ("layer1XDefOverride" 0.000)
        ("layer1YDefOverride" 0.032)
        ("layer1YEnclosure"   0.008)
        ("layer2XEnclosure"   0.008)
        ("layer2XDefOverride" 0.032)
        ("layer2YDefOverride" 0.000)
     )
     ( viaM2V2M3_LY2x2 viaM2V2M3
        ("row"    2)
        ("column" 2)
        ("xCutSpacing" 0.08)
        ("yCutSpacing" 0.08)
        ("layer1XDefOverride" 0.014)
        ("layer1YEnclosure"   0.014)
        ("layer2XEnclosure"   0.008)
        ("layer2YDefOverride" 0.008)
     )
) ;customViaVariants
```

Defines custom via variants named `viaM2V2M3_LY1x1` and `viaM2V2M3_LY2x2` of the custom via definition `viaM2V2M3` and adds a set of parameters to these.

cdsGenViaDefs

```
cdsGenViaDefs(
     ( t_viaDefName
       (layers
       ; ** Base Layers and extra layers **
          (layer1 tx_layer1)
          (layer2 tx_layer2)
          (cutLayer tx_cutLayer)
       )
       [(extraLayers
          ; ** Extra layers **
          [(layer1ExtraLayers l_layer1ExtraLayers)]
          [(layer2ExtraLayers l_layer2ExtraLayers)]
          [(cutExtraLayers l_cutExtraLayers)]
        )
       ]
       ; l_extraLayers := ({tx_layer} …)
       (parameters
          (cutWidth f_cutWidth)
          (cutHeight f_cutHeight)
          ; ** Other Default parameters **
          [(layer1Purpose tx_layer1Purpose)]
          [(layer1Enc l_layer1Enc)]
          [(layer2Purpose tx_layer2Purpose)]
          [(layer2Enc l_layer2Enc)]
          [(cutPurpose tx_cutPurpose)]
          [(cutSpacing x_cutSpacingX x_cutSpacingY)]
          [(cutRows n_cutRows)]
          [(cutColumns n_cutColumns)]
          [(cutPattern l_cutPattern)]
          [(alignment t_alignment)]
          [(originOffset (f_XOffset f_YOffset)]
          [(layer1ExtraParams l_layer1ExtraParams)]
          [(layer2ExtraParams l_layer2ExtraParams)]
          [(cutLayerExtraParams l_cutLayerExtraParams)]
          [(cutArraySpacing f_XCutArraySpacing f_YCutArraySpacing)]
          [(cutArrayPatternX l_cutArrayPatternX)]
          [(cutArrayPatternY l_cutArrayPatternY)]
          [(version n_version)]
       )
     ; l_enc := ( x_left x_right x_top x_bottom )
     ; l_pattern := ( ( {0|1} ... ) ... )
     ; l_extraLayerParams := ( ( [(enc l_enc)] [(purpose tx_purpose)] ) ... )
     ; l_cutArrayPattern := ( {x_num} ... )
     )
     …
) ;cdsGenViaDefs
```

Description

Specifies the definition of the layers and the default parameter values to use for all instances of the vias created from the definition.

It declares vias of the Cadence-defined `cdsGenVia` type with advanced features such as purpose-specific vias with an unlimited number of extra layers for each of the three primary layers and cut array support.

The via parameters can be grouped as follows:

- Parameters for *`layer1`*, which include purpose and enclosure
- Parameters for *`layer2`*, which include purpose and enclosure
- Parameters for the cut layer, which include purpose
- The cut width, height, and spacing
- The number of rows, columns, and optional pattern
- The via justification or origin offset
- An optional list of extra *`layer1`* layer parameters
- An optional list of extra *`layer2`* layer parameters
- An optional list of extra cut layer parameters
- An optional specification of how the cuts are to be grouped and distributed in arrays

Each of the three primary layers can have any number of extra layers. Shapes on the extra layers are generated relative to their parent layer, with an optional enclosure and shape purpose value.

Cuts can be grouped into arrays for multi-cut `cdsGenVias`. In such a case, the `cutSpacing` value applies to cuts within each array and `cutArraySpacing` specifies the spacing between arrays. The size of each array is specified using `cutArrayPatternX` and `cutArrayPatternY`. For example, if there are `7` cut rows, and `cutArrayPatternY` has the value `(3 4)`, then the cuts would be grouped into `2` arrays in the `Y` direction, the lower one with `3` cuts and the upper one with `4` cuts.

Arguments

`t_viaDefName`	The via definition name. Valid values: Any string; it must be unique in the technology database graph

`layer1` *`tx_layer1`*	The bottom routing layer of the via. Valid values: The layer name or the layer number
`layer2` *`tx_layer2`*	The top routing layer of the via. Valid Values: The layer name or the layer number
`cutLayer` *`tx_cutLayer`*	The cut layer. Valid values: The layer name or the layer number; the layer function must be `cut` **Note:** The mask number of the bottom layer must be lower than the mask number of the cut layer, which, in turn, must be lower than the mask number of the top layer.
`layer1ExtraLayers` *`l_layer1ExtraLayers`*	A list of layer names on which additional via shapes are generated. The shapes are generated with an enclosure relative to `layer1` as specified in `layer1ExtraParams`.
`layer2ExtraLayers` *`l_layer2ExtraLayers`*	A list of layer names on which additional via shapes are generated. The shapes are generated with an enclosure relative to `layer2` as specified in `layer2ExtraParams`.
`cutExtraLayers` *`l_cutExtraLayers`*	A list of layer names on which additional via shapes are generated. The shapes are generated with an enclosure relative to `cutLayer` as specified in `cutLayerExtraParams`.
`cutWidth` *`x_cutWidth`*	The width of a cut. Valid values: A floating-point number
`cutHeight` *`x_cutHeight`*	The height of a cut. Valid values: A floating-point number
`layer1Purpose` *`tx_layer1Purpose`*	The purpose name of the bottom routing layer-purpose pair.
`layer1Enc` *`l_layer1Enc`*	

A list defining the distance from the edge of the cut array to the edge of `layer1`. The enclosure is specified separately on all four sides. The syntax is as follows:
`(`*`x_left x_right x_top x_bottom`*`)`
Valid values: A floating-point number

`layer2Purpose` *`tx_purpose`*

The purpose name of the top routing layer-purpose pair.

`layer2Enc` *`l_layer2Enc`*

A list defining the distance from the edge of the cut array to the edge of `layer2`. The enclosure is specified separately on all four sides. The syntax is as follows:
`(`*`x_left x_right x_top x_bottom`*`)`
Valid values: A floating-point number

`cutPurpose` *`tx_cutPurpose`*

The purpose for cut shapes. The default value is `drawing`.

`cutSpacing` *`x_cutSpacingX x_cutSpacingY`*

A list defining the distance between cuts.The syntax is as follows:
`(` *`x_spaceX x_spaceY`* `)`
where, *`x_spaceX`* is the spacing between cuts in the X direction, and *`x_spaceY`* is the spacing between cuts in the Y direction.
Valid values: A floating-point number

`cutRows` *`n_cutRows`* The number of cut rows.
Valid values: An integer

`cutColumns` *`n_cutColumns`*

The number of cut columns.
Valid values: An integer

`cutPattern` *`l_cutPattern`*

A list representing the cuts in the via. To enable a cut, use `1`, and to disable a cut use `0`.

`alignment` *`t_alignment`*

The alignment of the cut matrix.
Valid values: `upperLeft`, `centerLeft`, `lowerLeft`, `upperCenter`, `centerCenter`, `lowerCenter`, `upperRight`, `centerRight`, `lowerRight`, `offset`

`originOffset (`*`f_XOffset f_YOffset`*`)`

The origin offset of the via as an absolute value. The syntax is is follows:
`(`*`x_offsetX x_offsetY`*`)`
This parameter can be used only if the alignment is set to `offset`.
Valid values: A floating-point number

`layer1ExtraParams` *`l_extraLayer1Params`*

The enclosure and purpose for the extra `layer1` layers. This is an ordered list with entries for each extra layer.
Valid values: `enc`, `purpose`

`layer2ExtraParams` *`l_extraLayer2Params`*

The enclosure and purpose for the extra `layer2` layers. This is an ordered list with entries for each extra layer.
Valid values: `enc`, `purpose`

`cutLayerExtraParams` *`l_cutLayerExtraParams`*

The enclosure and purpose for the extra `cut` layers. This is an ordered list with entries for each extra layer.
Valid values: `enc` and `purpose`

Note: If there are multiple rows or columns, a corresponding shape on each extra cut layer is generated for each cut shape. The enclosure is relative to the individual cut shape.

`cutArraySpacing (`*`f_XOffset f_YOffset`*`)`

The spacing between cut arrays, if cuts are grouped into arrays.

`cutArrayPatternX` *`l_cutArrayPatternX`*

Each entry specifies the number of cuts to group together in a left-to-right direction. The sum of all numbers must be equal to the number of cut columns.

`cutArrayPatternY` *`l_cutArrayPatternY`*	
	Each entry specifies the number of cuts to group together in a bottom-to-top direction.The sum of all numbers must be equal to the number of cut columns.
`version` *`n_version`*	The version number of the via generator to use. The only supported value is `1`. The version number is incremented by 1 whenever the generator is updated due to an extension or change in the parameter set.

Example

```
cdsGenViaDefs(
     (via01
       (layers
          (layer1   Poly)
          (layer2   Metal1)
          (cutLayer Cont)
       )
       (parameters
          (cutWidth   0.2)
          (cutHeight  0.2)
       )
     )
     (via1_Nwell_Nimp_hv
       (layers
          (layer1 "Metal1")
          (layer2 "Metal2")
          (cutLayer "Via1")
       )
       (extraLayers
          (layer1ExtraLayers ("Nwell" "Nimp"))
       )
       (parameters
          (cutWidth   0.2)
          (cutHeight  0.2)
          (cutSpacing 0.1 0.1)
          (cutRows    1)
          (cutColumns 1)
          (layer1Purpose "hv")
          (layer1Enc (0.3 0.3 0.0 0.0))
          (layer2Purpose "hv")
          (layer2Enc (0.1 0.1 0.2 0.2))
          (cutPurpose "hv")
          (alignment "centerCenter")
          (layer1ExtraParams (
               ( (purpose hv) (enc (0.1 0.1 0.1 0.1)) )  ; Nwell
               ( (purpose hv) (enc (1.0 1.0 0.0 0.0)) )  ; Nimp
         )
       )
     (via1_offset
       (layers
          (layer1 Metal1)
          (layer2 Metal2)
          (cutLayer Via1)
       )
       (parameters
          (cutWidth   0.2)
          (cutHeight  0.2)
          (cutSpacing 0.1 0.1)
          (cutRows    1)
          (cutColumns 1)
          (alignment "offset")
          (originOffset (0.1 0.2) )
       )
     )
     (via1_arrays
       (layers
          (layer1 Metal1)
          (layer2 Metal2)
          (cutLayer Via1)
       )
       (parameters
          (cutWidth   0.2)
          (cutHeight  0.2)
          (cutSpacing 0.1 0.1)
          (cutRows    5)
          (cutColumns 4)
          (alignment "lowerLeft")
          (cutArraySpacing  0.15 0.15 )
          (cutArrayPatternX (2 2) )
          (cutArrayPatternY (2 3) )
       )
     )
     (via1_pattern
       (layers
          (layer1 Metal1)
          (layer2 Metal2)
          (cutLayer Via1)
       )
       (parameters
          (cutWidth   0.2)
          (cutHeight  0.2)
          (cutSpacing 0.1 0.1)
          (cutRows    5)
          (cutColumns 4)
          (alignment "lowerLeft")
```

```
          (cutPattern ( (1 0 1 0 ) (0 1 0 1) (1 0 1 0) (0 1 0 1) (1 0 1 0) ) )
        )
      )

) ;cdsGenViaDefs
```

cdsGenViaVariants

```
cdsGenViaVariants(
     ( t_viaVariantName
       t_viaDefName
       (parameters
          [(layer1Purpose tx_layer1Purpose)]
          [(layer2Purpose tx_layer2Purpose)]
          [(cutPurpose tx_cutPurpose)]
          [(cutWidth f_cutWidth)]
          [(cutHeight f_cutHeight)]
          [(cutColumns n_cutColumns)]
          [(cutRows n_cutRows)]
          [(cutSpacing f_Xspacing f_Yspacing)]
          [(layer1Enc l_fourEnclosures1)]
          [(layer2Enc l_fourEnclosures2) ]
          [(cutPattern  l_cutPattern )]
          [(alignment t_alignment)]
          [(originOffset (f_Xoffset f_Yoffset) ]
          [(cutArraySpacing f_XCutArraySpacing f_YCutArraySpacing)]
          [(version n_version)]
          [(layer1ExtraParams l_layer1ExtraParams)]
          [(layer2ExtraParams l_layer2ExtraParams)]
          [(cutArrayPatternX l_cutArrayPatternX)]
          [(cutArrayPatternY l_cutArrayPatternY)]
       )
       )
     ) ;cdsGenViaVariants
```

Description

Specifies generated via variants and the default parameter values to use for all instances of the generated via variants created from a specified generated via definition.

Arguments

t_viaVariantName	The generated via variant name. Valid values: Any string; it must be unique in the technology database graph
t_viaDefName	The name of an existing generated via definition to associate with the variant.
`parameters`	The parameters for the variant. For descriptions of parameters, see cdsGenViaDefs.

Example

```
cdsGenViaVariants(
;(viaVariantName viaDefName (parameters (paramName paramValue) ...))
;( ----------------------------------------------------------------------- )
    ( vv1 myGenVia parameters
        (layer1Purpose "net")
        (layer1Enc (0.0 0.0 0.1 0.1))
        (layer2Purpose "net")
        (layer2Enc (0.2 0.2 0.2 0.2))
        (cutPurpose "fill")
        (cutWidth 0.1)
        (cutHeight 0.1)
        (cutSpacing 0.1 0.1)
        (cutRows 8)
        (cutColumns 6)
        (cutPattern ((0 1 1 1 1 1) (1 0 1 1 1 1) (1 1 1 1 1 1) (1 1 1 1 1 1)
            (1 1 1 1 1 1) (1 1 1 1 1 1) (1 1 1 1 1 1) (1 1 1 1 1 1)))
        (alignment "offset")
        (originOffset (0.1 0.1))
        (layer1ExtraParams (((purpose "net") (enc (0.3 0.3 0.3 0.3)))
            ((purpose "drawing") (enc (0.2 0.2 0.2 0.2)))))
        (layer2ExtraParams (((purpose "net") (enc (0.3 0.3 0.3 0.3)))))
        (cutLayerExtraParams (((purpose "net") (enc (0.3 0.3 0.3 0.3)))))
        (cutArraySpacing 0.1 0.2)
        (cutArrayPatternX (2 4))
        (cutArrayPatternY (3 5))
        (version 1)
    )
) ;cdsGenViaVariants
```

viaSpecs

```
viaSpecs(
     (
       tx_layer1
       tx_layer2
       lt_viaDefNames
       [
           ( n_layer1MinWidth
             n_layer1MaxWidth
             n_layer2MinWidth
             n_layer2MaxWidth
             lt_viaDefNames2
           )
       ]
     )
) ;viaSpecs
```

Description

Defines an array of via definitions and via variants, referred to as a via specification.

Note: `viaSpecs()` does not support `cdsGenViaDefs()`. The `oaViaSpec` class and corresponding infrastructure are obsolete and superseded by the `LEFSpecialRouteSpec` and the `oacValidRoutingVias` constraints it contains.

A via specification allows you to map two wire widths to a list of `viaDefNames`. It is used for power vias.

Earlier, via specifications were used to store only the LEF VIARULE statement in OpenAccess. However, in the IC releases that use OpenAccess kits supporting Data Model 4, the `validRoutingVias` constraint is used to make power vias symmetric with signal vias. In addition, to help applications find that constraint, it is placed in a constraint group named `LEFSpecialRouteSpec`.

For more information about the LEF VIARULE statement, see *LEF/DEF 5.8 Language Reference*.

Arguments

`tx_layer1`	The first layer associated with the via specification. Valid values: The layer name or the layer number
`tx_layer2`	The second layer associated with the via specification. Valid values: The layer name or the layer number
`lt_viaDefNames`	A list of the names of the via definitions in the via definition array associated with the via specification. The syntax is as follows: `(` *`t_viaDefName`* `… )` Valid values: Any valid via definition or via variant name
`n_layer1MinWidth`	The minimum width of the first layer associated with the via specification, if set. Valid values: Any number
`n_layer1MaxWidth`	The maximum width of the first layer associated with the via specification, if set. Valid values: Any number
`n_layer2MinWidth`	The minimum width of the second layer associated with the via specification, if set. Valid values: Any number
`n_layer2MaxWidth`	The maximum width of the second layer associated with the via specification, if set. Valid values: Any number
`lt_viaDefNames2`	A list of names of the via definitions in the via definition array to which the minimum and maximum widths apply. The syntax is as follows: `(` *`t_viaDefName`* `… )` Valid values: Any valid via definition name

Example

```
viaSpecs(
         ( METAL1 METAL2 ( "Via12Array" )
           (
             (0.5 0.6 0.4 0.5 ( "Via12Array_south" ))
             (0.6 0.7 0.4 0.5 ( "Via12Array_north" ))
           )
         )
) ;viaSpecs
```

8

Technology File Devices

This chapter contains the following topics pertaining to technology file devices:

devices

`devices()`

Description

Is the technology file devices section enclosure. It is used to activate the Cadence predefined device types, declare devices of those types, and specify multipart path templates.

After activating the Cadence predefined device classes with `tcCreateCDSDeviceClass`, you can specify the following in the `devices` section of the technology file:

- `cdsViaDevice` declares a `via` device.
- `ruleContactDevice` declares a `rule contact` device.
- `multipartPathTemplates` defines a template or a set of templates that specify relative object design (ROD) multipart paths (MPPs).
- `waveguideTemplates` defines a template or a set of templates that help generate waveguides.

Additionally, you can specify the following extract devices:

- `extractMOS`
- `extractRES`
- `extractCAP`
- `extractDIODE`

tcCreateDeviceClass

```
tcCreateDeviceClass(
     t_viewName
     t_className
     l_classParam
     l_formalParam
     geometry
) ;tcCreateDeviceClass
```

Description

Defines a custom device type. The class parameters define the physical characteristics of the device; for example layer-purpose pairs for source, drain, or gate and enclosure values. The formal parameters are those that you want to be able to modify when you place the device. The geometry portion of the device type definition is SKILL code that defines the rectangles, lines, nets, terminals, and pins that make up the device type.

Important

This subclass must be included before any `tcDeclareDevice` statements that use the device type. Moreover, you must use a new `tcCreateDeviceClass` subclass for every custom device type that you create.

Arguments

t_viewName	The name of the view you want devices of this type to have.
t_className	The name of the device class to define.

`l_classParam`	A list of the class parameters you want the device to have. Class parameters are user-definable. This argument has the following syntax: `( (` *`t_parameter g_value`* `) … )` where, ■ *`t_parameter`* is the name of the class parameter. Valid values: Any string ■ *`g_value`* is a default for the parameter. The purpose of this value is to identify the type of value you can use. When you declare a device, you set the value to be used for the device. Valid values: An integer, a floating-point number, a string enclosed in quotation marks, a Boolean value, or any SKILL symbol or expression that evaluates to any of these types.
`l_formalParam`	A list of the formal parameters you want to be able to modify while or after the device has been placed. Formal parameters are user-definable. This argument has the following syntax: `( (` *`t_parameter g_value`* `) … )` ■ *`t_parameter`* is the name of the formal parameter. Valid values: Any string ■ *`g_value`* is a default for the parameter. The purpose of this value is to identify the type of value you can use. When you declare a device, you set the value to be used for the device. Valid values: An integer, a floating-point number, a string enclosed in quotation marks, a Boolean value, or any SKILL symbol or expression that evaluates to any of these types.
`geometry`	A list of the SKILL functions that specify how the class and formal parameters are used to create the device.

Example 1

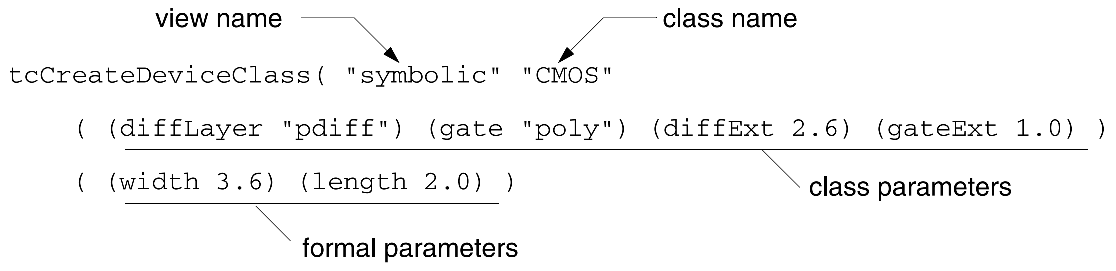
```
; Create "syMGEnhancement" device class
;
tcCreateDeviceClass("symbolic" "syMGEnhancement"
; class parameters
   ( (sdLayer "hilite") (gateLayer "hilite")
     (sdExt 0.0) (gateExt 0.0)
     (sdImpLayer nil) (sdImpEnc 0.0) )
; formal parameters
   ( (width 0.0) (length 0.0) )
; geometry
    W2 = width/2  L2 = length/2
    netId = dbMakeNet(tcCellView "G")
    dbId = dbCreateDot(tcCellView gateLayer -W2-gateExt:0)
    dbId = dbCreatePin(netId dbId "gl")
    dbSetq(dbId list("left") accessDir)
    dbId = dbCreateDot(tcCellView gateLayer W2+gateExt:0)
    dbId = dbCreatePin(netId dbId "gr")
    dbSetq(dbId list("right") accessDir)
    dbId = dbCreateRect(tcCellView gateLayer
        list(-W2-gateExt:-L2 W2+gateExt:L2))
    dbAddFigToNet(dbId netId)
    ;
    netId = dbMakeNet(tcCellView "S")
    dbId = dbCreateDot(tcCellView sdLayer 0:L2)
    dbId = dbCreatePin(netId dbId "s")
    dbSetq(dbId list("top") accessDir)
    dbId = dbCreateRect(tcCellView sdLayer list(-W2:0 W2:L2+sdExt))
    dbAddFigToNet(dbId netId)
) ; end of syMGEnhancement
```

Shows a customized device type. When you declare a device of this type with `tcDeclareDevice`, you specify the values used in a device.

Example 2

Creates a class called `CMOS`. The SKILL functions are described in small groups. The entire device class definition appears at the end of this example.

The first line in the following example names the class `CMOS` and assigns it the view name `symbolic` (that is, all devices in this class are saved as symbolic cellviews). The closing parenthesis for the `tcCreateDeviceClass` statement appears at the end of the entire device class definition.



```
tcCreateDeviceClass( "symbolic" "CMOS"

    ( (diffLayer "pdiff") (gate "poly") (diffExt 2.6) (gateExt 1.0) )

    ( (width 3.6) (length 2.0) )
```

The second line in the figure above lists the class parameters. You can create any parameter name you want and assign it a default value. In this example, the class parameters `diffLayer` and `gate` assign the layers on which the device is constructed. The `diffExt` and `gateExt` parameters define the amount of extension on the `diffusion` and `gate` layers, in microns.

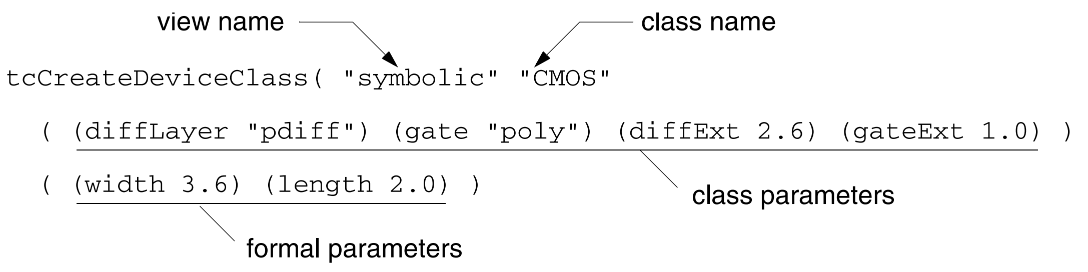
The third line in the figure above lists the formal parameters. Again, you can create any parameter name you wish. The formal parameters appear in the options window when you place instances for this class. You should assign the parameters meaningful names that make sense to the person placing the device. In this example, the formal parameters `width` and `length` assign the width and length of the transistor (the area where `diffusion` and `gate` intersect), in microns.

The formal and class parameters are followed by the SKILL code that describes how to construct the device. You can use the SKILL database access commands (`db` commands) for describing database objects to define the rectangles, lines, nets, terminals, and pins required by each device in the class.

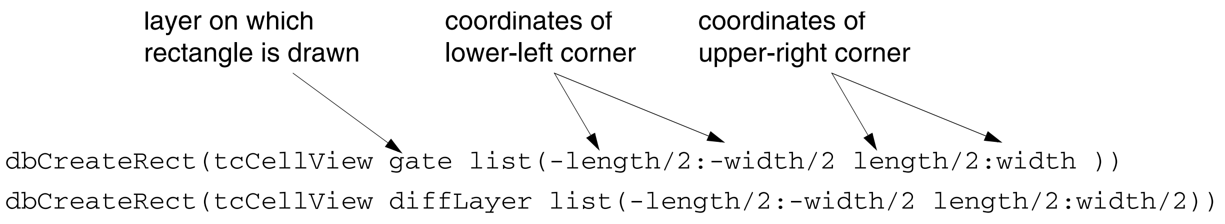
The first line in the following example defines a rectangle on the gate layer. The second line defines a rectangle of the same size on the `diffusion` layer. Together, these two rectangles make up the active area of the transistor.

In this example, `tcCellView` is a previously defined global variable that assigns the view name, `symbolic`. The global variable is defined only while the SKILL code is being run. The remaining four lines define the `diffusion` and `gate` extensions, assigning variable names (`gu`, `gl`, `ss`, and `dd`) to some of the rectangles for use later in this device class definition.

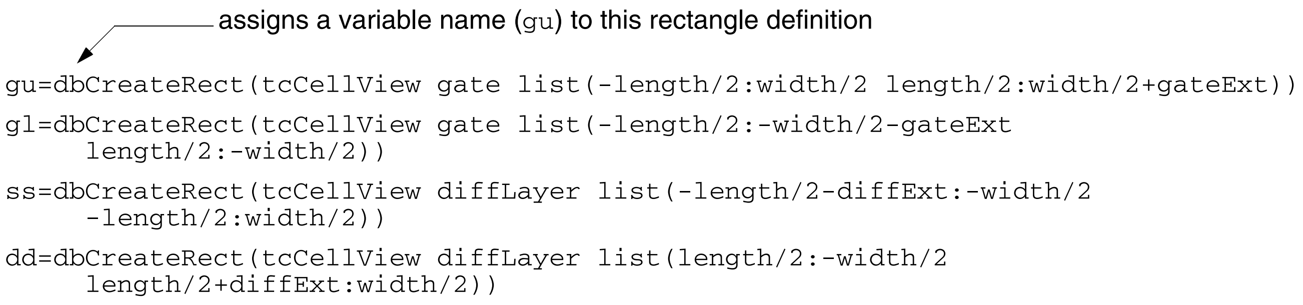
Each succeeding `dbCreateRect` function call creates a new device variant, a rectangle with `tcCellView` as the master cellview of the variant. All of the rectangles are defined in relation to the origin (`0,0`) of the device. For example, given the default `length` and `width` values of `2.0` and `3.6`, the coordinates of the first rectangle are (`-1`, `-1.8`) and (`1`, `1.8`). The resulting rectangle is `2.0` microns wide (the `length` value) by `3.6` microns high (the `width` value) and is centered around the origin of the transistor.



```
tcCreateDeviceClass( "symbolic" "CMOS"
  ( (diffLayer "pdiff") (gate "poly") (diffExt 2.6) (gateExt 1.0) )
  ( (width 3.6) (length 2.0) )
```



```
dbCreateRect(tcCellView gate list(-length/2:-width/2 length/2:width ))
dbCreateRect(tcCellView diffLayer list(-length/2:-width/2 length/2:width/2))
```



```
gu=dbCreateRect(tcCellView gate list(-length/2:width/2 length/2:width/2+gateExt))
gl=dbCreateRect(tcCellView gate list(-length/2:-width/2-gateExt
     length/2:-width/2))
ss=dbCreateRect(tcCellView diffLayer list(-length/2-diffExt:-width/2
     -length/2:width/2))
dd=dbCreateRect(tcCellView diffLayer list(length/2:-width/2
     length/2+diffExt:width/2))
```

The final section of the SKILL code defines the nets, terminals, and pins required by this device. The first line in the following example defines a new net named `g` and assigns that net the variable name `gNet`. The next three lines define one terminal and two pins on this net.

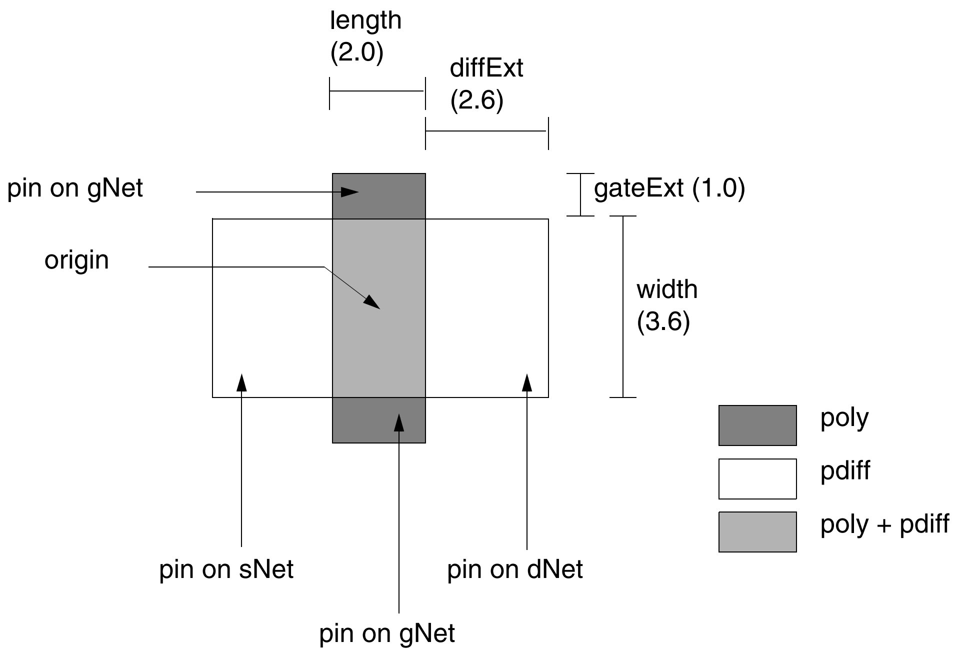
- The terminal name is `g` and its direction is `input`.
- The first pin is assigned to the rectangle named `gu` and given the name `u`.
- The second pin is assigned to the rectangle named `gl` and given the name `l`.

The last lines define two more nets with a terminal and a pin each. The final parenthesis closes this device class definition.

assigns a variable name (`gNet`) to this net

```
gNet=dbMakeNet(tcCellView "g")            creates a net named "g"

dbCreateTerm(gNet "g" "input")            creates an input terminal named "g" on gNet

dbCreatePin(gNet gu "u")                  creates a pin named "u" for gNet on rectangle gu

dbCreatePin(gNet gl "l")                  creates a pin named "l" for gNet on rectangle gl

sNet=dbMakeNet(tcCellView "s")
dbCreateTerm(sNet "s" "inputOutput")
dbCreatePin(sNet ss)
dNet=dbMakeNet(tcCellView "d")
dbCreateTerm(dNet "d" "inputOutput")
dbCreatePin(dNet dd)
)
```

The resulting default transistor for this class appears as follows:



You can change the dimensions and layers for the individual devices in this class as you define devices in the technology file.

The entire set of SKILL functions for this device class is repeated below:

```
;Section 1
tcCreateDeviceClass(  "symbolic" "CMOS"
  ( (diffLayer "pdiff") (gate "poly")
    (diffExt 2.6) (gateExt 1.0) )
  ( (width 3.6) (length 2.0) )
;
;Section 2
;draw the rectangles for the device
dbCreateRect(tcCellView gate list(-length/2:-width/2 length/2:width/2))
dbCreateRect(tcCellView diffLayer list(-length/2:-width/2 length/2:width/2))
gu=dbCreateRect(tcCellView gate list(-length/2:width/2
     length/2:width/2+gateExt))
gl=dbCreateRect(tcCellView gate list(-length/2:-width/2-gateExt
     length/2:-width/2))
ss=dbCreateRect(tcCellView diffLayer list(-length/2-diffExt:-width/2
     -length/2:width/2))
dd=dbCreateRect(tcCellView diffLayer list(length/2:-width/2
     length/2+diffExt:width/2))
;
;Section 3
;Define nets, pins, and terminals
;Assign pins to previously defined rectangles
gNet=dbMakeNet(tcCellView "g")
dbCreateTerm(gNet "g" "inputOutput")
dbCreatePin(gNet gu "u")
dbCreatePin(gNet gl "l")
sNet=dbMakeNet(tcCellView "s")
dbCreateTerm(sNet "s" "inputOutput")
dbCreatePin(sNet ss)
dNet=dbMakeNet(tcCellView "d")
dbCreateTerm(dNet "d" "inputOutput")
dbCreatePin(dNet dd)
)
```

tcDeclareDevice

```
tcDeclareDevice(
     t_viewName
     t_className
     t_deviceName
     (('s_classParam g_value) …)
     (('s_formalParam g_value) …)
=> t / nil
```

Description

Creates the device defined by `tcCreateDeviceClass`.

Arguments

t_viewName	The name of the view you want devices of this type to have.
t_className	The name of the device class to be defined.
t_deviceName	The name of the device to be defined.
s_classParam	A list of the class parameters you want the device to have. Class parameters are user-definable. This argument has the following syntax: `( ( s_parameter g_value ) … )` ■ *s_parameter* is the name of the class parameter. Valid values: Any string ■ *g_value* is a default for the parameter. The purpose of this value is to identify the type of value you can use. When you declare a device, you set the value to be used for the device. Valid values: An integer, a floating-point number, a string enclosed in quotation marks, a Boolean value, or any SKILL symbol or expression that evaluates to any of these types.

s_formalParam	A list of the formal parameters you want to be able to modify while or after the device has been placed. Formal parameters are user-definable. This argument has the following syntax: `( ( s_parameter g_value ) … )` ■ *s_parameter* is the name of the formal parameter. Valid values: Any string ■ *g_value* is a default for the parameter. The purpose of this value is to identify the type of value you can use. When you declare a device, you set the value to be used for the device. Valid values: an integer, a floating-point number, a string enclosed in quotation marks, a Boolean value, or any SKILL symbol or expression that evaluates to any of these types.

Example

```
tcDeclareDevice( "symbolic" "syEnhContact" "myEnhCont"
    ( (viaLayer "Via1") (layer2ImpEnc 0.000000) (layer2ImpLayer (quote nil))
    (layer1ImpEnc 0.450000) (layer1ImpLayer "Nimp") (encByLayer2 0.350000)
    (encByLayer1 0.310000) (layer2 "Metal2") (layer1 "Metal1") )
    ( (w 0.300000) (myProp2 "net2") (myProp1 "net1")
    (yBias "center") (xBias "center") (yPitch 0.800000)
    (xPitch 0.700000) (column 1) (row 2)(l 0.300000) )
)
```

tcDeleteDevice

```
tcDeleteDevice(
     t_viewName
     t_className
     t_deviceName
     [d_techfileID]
)
```

Description

Deletes a device defined using `tcCreateDeviceClass` and `tcDeclareDevice`. This subclass must be placed in the `devices` class of the technology file after the `tcCreateDeviceClass` and `tcDeclareDevice` declarations of the device being deleted.

Arguments

t_viewName	The view of the device.
t_className	The device class.
t_deviceName	The name of the device.
d_techfileID	Do not use in ASCII techfiles.

tfcDefineDeviceProp

```
tfcDefineDeviceProp(
      (viewName deviceName propName propValue)
      …
)
=> t/nil
```

Description

Defines the device property.

Arguments

viewName	The name of the view you want devices of this type to have.
deviceName	The name of the device to be defined.
propName	The name of the property to be defined.
propValue	The value of the property to be defined.

Example

```
tfcDefineDeviceProp(
     (symbolic POLYS_M1 res      1.000000 )
     (symbolic M1_M2    res      1.500000 )
     (symbolic M3_M2    viaType  "default")
)
```

tcCreateCDSDeviceClass

```
tcCreateCDSDeviceClass()
```

Description

Activates the Cadence predefined device classes—`cdsViaDevice` and `ruleContactDevice`. You can also specify `multipartPathTemplates`. After the Cadence predefined device classes are activated, you can declare specific devices of these types with the appropriate device specification.

You must place `tcCreateCDSDeviceClass()` in the `devices` section before you define any devices by using the Cadence predefined device types.

Example

```
tcCreateCDSDeviceClass()
```

Activates the Cadence predefined device types.

cdsViaDevice

```
cdsViaDevice(
     (t_deviceName t_cutLayer t_cutPurpose
     t_layer1 t_layer1Purpose
     t_layer2 t_layer2Purpose
     n_row n_column t_origin g_stackedVias
     g_cutLayerW g_cutLayerL g_xCutSpacing g_yCutSpacing
     g_layer1XDirOverride g_layer1YDirOverride
     g_layer2XDirOverride g_layer2YDirOverride
     t_layer1Dir t_layer2Dir
     g_layer1XDefOverride g_layer1YDefOverride
     g_layer2XDefOverride g_layer2YDefOverride
     l_implantLayer1
     l_implantLayer2
     g_diffSpacing
     t_abutClass
     )
     …
) ;cdsViaDevice
```

Description

Declares devices of the Cadence predefined `via` type. A via device is a single contact or array of contacts placed between two layers. This section must be included after `tcCreateCDSDeviceClass` in the `devices` section of the technology file.

A `cdsViaDevice`, rather than a `standardViaDef`, is required when one or more of the following conditions exist:

- The via has multiple `layer1` and/or `layer2` implant layers.
- The spacing between the diffusion layers of two vias that cannot abut (*`g_diffSpacing`*) must be specified.
- The via must be assigned an abutment class (*`t_abutClass`*).

Important

We recommend that you use `standardViaDefs`. However, if you cannot get your layout configuration by using `standardViaDefs`, then you should use `cdsViaDevice`.

You can specify `cdsViaDevice` directly in the technology file, or when you create a substrate tie, the technology file compiler automatically creates `cdsViaDevice` in the technology file. In either case, the compiler also creates a custom via definition that references the device. If you specify `cdsViaDevice` when none of the conditions listed above exist and `techVersion` is not used, the compiler creates a standard via definition and does not create the device.

Important

Three formal parameters, `primaryShapePurpose`, `cutShapePurpose`, and `otherShapePurpose`, with default value `""` (an empty string), can be used to override the purpose when creating a cdsVia or while updating the parameters of an existing cdsVia. However, these parameters are not specified in the ASCII technology file.

The `primaryShapePurpose` parameter allows specification of `layer1Purpose` and `layer2Purpose`, the `cutShapePurpose` parameter allows specification of `cutPurpose`, and the `otherShapePurpose` parameter allows specification of the purpose for the `implantLayer1` and `implantLayer2` implant layers for cdsVias. If the value `""` is used for any of these parameters, the associated purpose or purposes take the default value specified in the ASCII technology file for that cdsVia.

Let us consider a scenario where you have specified the primary purpose `hv` for a cdsVia between layers `m1` and `m2`. If `<m2:hv>` is not a valid LPP, the default LPP specified in the technology file is considered.

Arguments

`t_deviceName`	The name of the via device to declare.
`t_cutLayer`	The name of the via layer to use.
`t_cutPurpose`	The name of the via purpose to use.
`t_layer1`	The layer name of the bottom routing LPP.
`t_layer1Purpose`	The purpose name of the bottom routing LPP.
`t_layer2`	The layer name of the top routing LPP.
`t_layer2Purpose`	The purpose name of the top routing LPP.
`n_row`	The number of rows of contact cuts in a contact array.
`n_column`	The number of columns of contact cuts in a contact array.
`t_origin`	The origin of the contact or via cut. Valid values: `lowerLeft`, `centerLeft`, `upperLeft`, `lowerCenter`, `centerCenter`, `upperCenter`, `lowerRight`, `centerRight`, `upperRight`

g_stackedVias	Indicates whether vias can be stacked. Valid values: `t`, `nil`
g_cutLayerW	Specifies the dimension of the cut in X direction. Valid values: A floating-point number or `_def_` (`_def_` uses the `minWidth` spacing rule)
g_cutLayerL	Specifies the dimension of the cut in Y direction. Valid values: A floating-point number or `_def_` (`_def_` uses the `minWidth` spacing rule)
g_xCutSpacing	The distance, edge-to-edge, between horizontal cuts in a contact array. Valid values: A floating-point number or `_def_` (`_def_` uses the `minSpacing` spacing rule)
g_yCutSpacing	The distance, edge-to-edge, between vertical cuts in a contact array. Valid values: A floating-point number or `_def_` (`_def_` uses the `minSpacing` spacing rule)
g_layer1XDirOverride	Minimum enclosure for `layer1` and the cut layer in the X direction. Default: *g_layer1XDefOverride*.
g_layer1YDirOverride	Minimum enclosure for `layer1` and the cut layer in the Y direction. Default: *g_layer1YDefOverride*.
g_layer2XDirOverride	Minimum enclosure for `layer2` and the cut layer in the X direction. Default: `g_layer2XDefOverride`.
g_layer2YDirOverride	Minimum enclosure for `layer2` and the cut layer in the Y direction. Default: `g_layer2YDefOverride`

t_layer1Dir	Sets the direction from `layer1`. Valid values: `t`, `b`, `l`, `r`, `x`, `y`, `none`, where each valid value is interpreted as follows: `t`: top `b`: bottom `l`: left `r`: right `x`: left and right `y`: top and bottom `none`: none Default: An empty string
t_layer2Dir	Sets the direction from `layer2`. Valid values: `t`, `b`, `l`, `r`, `x`, `y`, `none`, where each valid value is interpreted as follows: `t`: top `b`: bottom `l`: left `r`: right `x`: left and right `y`: top and bottom `none`: none Default: An empty string
g_layer1XDefOverride	Minimum enclosure for `layer1` and the cut layer in the X direction for the directions not specified in *t_layer1Dir*.
g_layer1YDefOverride	Minimum enclosure for `layer1` and the cut layer in the Y direction for the directions not specified in *t_layer1Dir*.
g_layer2XDefOverride	Minimum enclosure for layer2 and the cut layer in the X direction for the directions not specified in *t_layer1Dir*.
g_layer2YDefOverride	Minimum enclosure for layer2 and the cut layer in the Y direction for the directions not specified in *t_layer1Dir*.

l_implantLayer1	A list defining the implant layers for `layer1`. The list has the following syntax:
	`(((`*`t_implantLayer t_implantPurpose`*`)`
	   `g_implantEnc`
	 `)`
	 …
	`)`
	where,
	■ *`t_implantLayer`* is the name of the implant layer.
	■ *`t_implantPurpose`* is the name of the implant purpose.
	■ *`g_implantEnc`* is the minimum spacing of the implant enclosure around `layer1`.
	Note: You can specify as many implant layers as needed.

l_implantLayer2	A list defining the implant layers for layer2. The list has the following syntax:
	`(((`*`t_implantLayer t_implantPurpose`*`)`
	   `g_implantEnc`
	 `)`
	 …
	`)`
	where,
	■ *`t_implantLayer`* is the name of the implant layer.
	■ *`t_implantPurpose`* is the name of the implant purpose.
	■ *`g_implantEnc`* is the minimum spacing of the implant enclosure around `layer2`.
	Note: You can specify as many implant layers as needed.

g_diffSpacing	The minimum spacing between the diffusion layers of two vias that cannot abut. Valid values: A floating-point number or `_def_` (`_def_` uses the `minSpacing` spacing rule)

`t_abutClass`	The abutment class to which the via belongs. Commonly used to enable abutment of a substrate to a MOS device by assigning each the same abutment class. Devices with the same abutment class abut. For more information about abutment, see Virtuoso Layout Suite documentation. Valid values: Any string; an empty string when no abutment class is assigned

Note: If a value is not specified, the `DirOverride` and `DefOverride` parameters look for the `minOppExtension` constraint. If the `minOppExtension` constraint is not found, then the `minExtensionDistance` constraint is looked up.

Example

```
cdsViaDevices(
    (M2_M1cds Via1 drawing Metal1 drawing Metal2 drawing
        1 1 centerCenter t _def_ _def_ _def_ _def_
        0 0 0 0
        "" ""
        0.005 _def_ 0.005 _def_
        nil
        nil
        _def_ ""
    )
    (M3_M2cds Via2 drawing Metal2 drawing Metal3 drawing
        1 1 centerCenter t _def_ _def_ _def_ _def_
        0 0 0 0
        "" ""
        0.005 _def_ 0.005 _def_
        nil
        nil
        _def_ ""
    )
    (M4_M3cds Via3 drawing Metal3 drawing Metal4 drawing
        1 1 centerCenter t _def_ _def_ _def_ _def_
        0 0 0 0
        "" ""
        0.005 _def_ 0.005 _def_
        nil
        nil
        _def_ ""
    )
) ;cdsViaDevices
```

Defines cdsViaDevices. If `techVersion` is not specified, the cdsVias are converted to standardViaDefs. Otherwise, the cdsViaDevices are retained as cdsVias.

ruleContactDevice

```
ruleContactDevice(
     ( t_deviceName
       {( t_layer1 t_purpose1 l_rectangles ) | nil}
       [( t_viaLayer t_viaPurpose l_rectangles )]
       [( t_layer2 t_purpose2 l_rectangles )]
     )
     …
) ;ruleContactDevice
```

Description

Declares devices of the Cadence predefined `contact` type. A rule contact device is a single contact or a rectangular array of contacts placed between multiple (three or more) layers. This section must be included after `tcCreateCDSDeviceClass` in the `devices` section of the technology file.

The `ruleContactDevice` section in OA technology database is not a separate device class, but a method for custom via creation. The `customViaDefs` section does not create any device. It only lists cellviews, which should be created elsewhere.

For example:

```
;;; DEFINE THAT CELLVIEW via1_horz_test/symbolic IS A CUSTOM VIA
customViaDefs(
    ( via1_horz_test tsmc18_ams via1_horz_test symbolic m1 m2 6.4 )
) ; customViaDefs
;;; CREATE THE CELLVIEW via1_horz_test/symbolic (GEOM INFO IS GIVEN HERE)
devices(
    tcCreateCDSDeviceClass() ;init default devices
    ruleContactDevice(
        ( via1_horz_test
            ( m1   drawing ( -0.190000 -0.140000 0.190000 0.140000 ) )
            ( via1 drawing ( -0.130000 -0.130000 0.130000 0.130000 ) )
            ( m2   drawing ( -0.140000 -0.140000 0.140000 0.140000 ) )
        )
    );ruleContactDevice
) ; devices
```

Arguments

t_deviceName	The name of the rule contact device to declare.

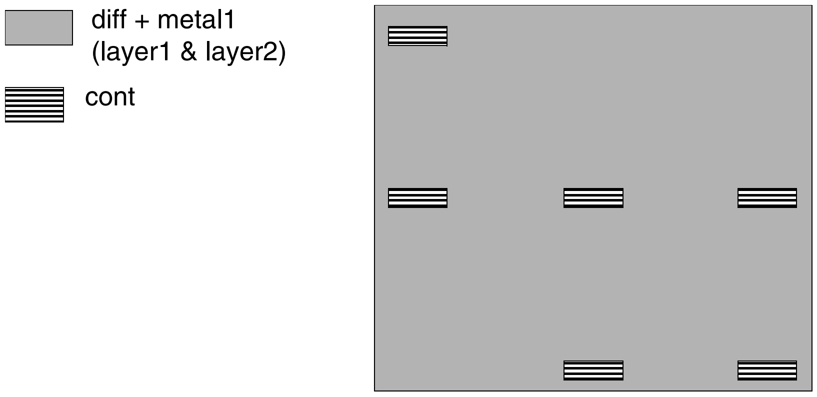
`t_layer1`	The layer name of the bottom routing layer. **Note:** To create a single layer rule contact device, specify `nil` for this argument.
`t_purpose1`	The purpose name of the bottom routing layer-purpose pair. **Note:** To create a single layer rule contact device, specify `nil` for this argument.
`l_rectangles`	A list of coordinates for rectangles to place on the bottom routing layer. The list has the following syntax: `( (` *`n_llX n_llY n_urX n_urY`* `) … )` where, ■ *`n_llX`* is the X coordinate for the lower-left corner of the rectangle. ■ *`n_llY`* is the Y coordinate for the lower-left corner of the rectangle. ■ *`n_urX`* is the X coordinate for the upper-right corner of the rectangle. ■ *`n_urY`* is the Y coordinate for the upper-right corner of the rectangle. **Note:** To create a single layer rule contact device, specify `nil` for this argument.
`t_viaLayer`	The name of the via layer to use.
`t_viaPurpose`	The purpose name of the via layer-purpose pair to use.

l_rectangles	A list of coordinates for rectangles to place on the via layer. The list has the following syntax: `( ( n_llX n_llY n_urX n_urY ) … )` where, ■ *n_llX* is the X coordinate for the lower-left corner of the rectangle. ■ *n_llY* is the Y coordinate for the lower-left corner of the rectangle. ■ *n_urX* is the X coordinate for the upper-right corner of the rectangle. ■ *n_urY* is the Y coordinate for the upper-right corner of the rectangle.
t_layer2	The name of the top routing layer to use.
t_purpose2	The purpose name of the top routing layer-purpose pair to use.
l_rectangles	List of a list of coordinates for rectangles to place on the top routing layer. The list has the following syntax: `(  (  n_llX n_llY n_urX n_urY ) … )` where, ■ *n_llX* is the X coordinate for the lower-left corner of the rectangle. ■ *n_llY* is the Y coordinate for the lower-left corner of the rectangle. ■ *n_urX* is the X coordinate for the upper-right corner of the rectangle. ■ *n_urY* is the Y coordinate for the upper-right corner of the rectangle.

Example

```
ruleContactDevice(
;( deviceName
 ( "VIABIGPOWER12"

;( layer1   purpose1   rectangles ) | nil
 ( diff     drawing    (-21.000 -21.000 21.000 21.000 ) )

;[( viaLayer   viaPurpose   rectangles )]
  ( cont       drawing      (  -2.400   -0.800    2.400     0.800   )
                            ( -19.000   17.400    -14.200   19.000  )
                            (  14.200   -19.000   19.000    -17.400 )
                            ( -19.000   -0.800    -14.200   0.800   )
                            (  -2.400   -19.000   2.400     -17.400 )
                            (  14.200   -0.800    19.000    0.800   )
  )

;[( layer2  purpose2   rectangles )]
  ( metal1   drawing    ( -21.000  -21.000  21.000  21.000 ) )
 )
) ; end of ruleContactDevice
```

Specifies the rule contact definition for a device called `VIABIGPOWER12`, which looks like the device shown in the figure below:



multipartPathTemplates

```
multipartPathTemplates(
     ( t_mppTemplateName
       l_template
     ) ; end of template
     …
     ) ;end of all multipartPathTemplates Rules

     ; l_template arguments
     (
         l_masterPathArgs
         [l_offsetSubpathArgs...]
         [l_enclosureSubpathArgs...]
         [l_subrectangleArgs...]
     ) ; end of template argument lists

     ; l_masterPathArgs
     (
         txl_layer
         [n_width]
         [g_choppable]
         [t_endType]
         [n_beginExt]
         [n_endExt]
         [t_justification]
         [n_offset]
         [l_rodConnectivityArgs for master path]
     ) ;end of masterPathArgs list

     ; l_offsetSubpathArgs
     (
          (
               txl_layer
               [n_width]
               [g_choppable]
               [n_sep]
               [t_justification]
               [n_beginOffset]
               [n_endOffset]
               [l_rodConnectivityArgs for offset subpath]
          ) ; end of first offset subpath list
          …
     ) ; end of all offset subpath lists

     ; l_enclosureSubpathArgs
     (
          (
               txl_layer
               [n_enclosure]
```

```
                [g_choppable]
                [n_beginOffset]
                [n_endOffset]
                [l_rodConnectivityArgs for enclosure subpath]
           ) ; end of first enclosure subpath list
           …
      ) ; end of all enclosure subpath lists

      ; l_subRectangleArgs
      (
           (
                txl_layer
                [n_width]
                [n_length]
                [g_choppable]
                [n_sep]
                [t_justification]
                [n_space]
                [n_beginOffset]
                [n_endOffset]
                [n_gap]
                [l_rodConnectivityArgs for subrectangles]
                [n_beginSegmentOffset]
                [n_endSegmentOffset]
           ) ; end of first subrectangle list
           …
      ) ; end of all subrectangle lists

      ; l_rodConnectivityArgs
      (
           [t_termIOType]
           [g_pin]
           [tl_pinAccessDir]
           [g_pinLabel]
           [n_pinLabelHeight]
           [txl_pinLabelLayer]
           [t_pinLabelJust]
           [t_pinLabelFont]
           [g_pinLabelDrafting]
           [t_pinLabelOrient]
           [t_pinLabelRefHandle]
           [l_pinLabelOffsetPoint]
      ) ; end of ROD Connectivity Argument list
```

Description

Defines a template or a series of templates that specify relative object design (ROD) multipart paths (MPPs). A multipart path is a single ROD object consisting of one or more parts at level zero in the hierarchy on the same or on different layers. An MPP template lets you create

MPPs in the layout by using predefined values from the technology file. You can define any number of MPP templates in this section; each template must be identified by a unique template name (`t_mppTemplateName`).

Arguments

`t_mppTemplateName`	Character string enclosed in double quotation marks, specifying the name of an MPP template. The name must be unique within the `multipartPathTemplates` section. Do not assign the name `New`; it is a reserved name. Default: `none`
`l_template`	List defining the MPP template. Valid values: A list of arguments

All other arguments for MPP templates in the `multipartPathTemplates` section have the same names, definitions, valid values, and default values as the arguments for the `rodCreatPath` function. For a detailed description of each argument, see the argument descriptions of the `rodCreatePath` function in *Virtuoso Relative Object Design SKILL Reference*.

Differences in the Syntax of multipartPathTemplates and rodCreatePath

The syntax for `multipartPathTemplates` is based on the syntax of the `rodCreatePath` function. For the arguments that `multipartPathTemplates` and `rodCreatePath` have in common, the argument definitions, valid values, and default values are identical.

The differences between the two are summarized below:

- Arguments for `multipartPathTemplates` are positional; you must specify them in the sequence shown in the syntax for `multipartPathTemplates`. Arguments for `rodCreatePath` are keyword value pairs; you can specify them in any sequence.
- There is one additional argument in `multipartPathTemplates`, *`t_mppTemplateName`*.
- For `multipartPathTemplates`, all arguments other than *`t_mppTemplateName`* are specified as lists within the *`l_template`* list, including the connectivity arguments.
- For some arguments, the data type is more restricted for `multipartPathTemplates` than it is for `rodCreatePath`. Specifically, you can enter either a character string or a symbol for `rodCreatePath` arguments with the data type *`S_`*; for the equivalent `multipartPathTemplates` arguments, you must enter a character string (the data type is *`t_`* for text).

- `multipartPathTemplates` does not contain the following `rodCreatePath` arguments because the values of these arguments vary for each occurrence of an MPP within a cellview:

  `S_name` and *`S_netName`*
  Enter values for these arguments in the *ROD Name* and *Net Name* fields in the Create ROD Multipart Path form.

  `S_termName`
  The value of *Net Name* is used for terminal name.

  `l_pts`
  Click in the layout cellview to specify the point list.

- `multipartPathTemplates` does not contain the `rodCreatePath` arguments listed below because these arguments represent an alternate way of specifying a point list for an MPP, and the point list varies for each occurrence of an MPP within a cellview:

  ```
  dl_fromObj     txf_size
  l_startHandle  l_endHandle
  ```

For more information about the `rodCreatePath` function, its arguments, and default values, see `rodCreatePath` in *Virtuoso Relative Object Design SKILL Reference*.

Defining, Changing, and Deleting MPP Templates

You can define new MPP templates, change the values for existing template definitions, and delete existing template definitions.

- You can **define** new MPPs for the `multipartPathTemplates` section by using any of the following three methods:
  - ❑ Define an MPP in the Create Multipart Path form and save it as a template. To access the form, choose *Create – Multipart Path* in VLS XL.

    For information about creating MPPs by using the graphical user interface, see Creating and Editing Multipart Paths in Virtuoso Layout Suite documentation.
  - ❑ Dump the ASCII version of your technology file, edit it, and then reload it in *Merge* or *Replace* mode.

    For information about loading the ASCII version of your technology file in *Replace* mode, see Replacing Existing Technology Data in a Technology Library in *Virtuoso Technology Data User Guide*.
  - ❑ To avoid the need to reload your technology file, use the `techSetMPPTemplate` function.

- You can **change** the contents of an existing MPP template definition by editing the template in the Create Multipart Path form. You can also edit the `multipartPathTemplates` section in the ASCII version of your technology file and reload it in *Merge* or *Replace* mode.
- You can **delete** an MPP template definition from the `multipartPathTemplates` section of your technology library. After editing the ASCII version of your technology file, you can reload it in *Replace* mode. You can also choose to specify the `techSetMPPTemplate` function as follows:

  ```
  techSetMPPTemplate( tfId "template_name" nil )
  ```

  where `tfId` is the technology file ID.

Note: All methods listed above affect only the temporary version of your technology library in virtual memory. If you want your changes to persist beyond the end of the current editing session, you must save the changes to the binary technology library on disk before you exit the software.

Specifying Positional Arguments

The arguments for defining an MPP template are positional. For example, if you want to specify values for only the first, fifth, and sixth arguments, you must specify `nil` or the required value for the second, third, and fourth arguments. You do not need to specify the arguments that appear after the sixth argument in the syntax.

The syntax is organized so that arguments you are most likely to specify are listed first and the arguments for which you probably want to use the default values are listed last. Most of the arguments have a system-assigned default value and are, therefore, optional. The only required argument is *`tx1_layer`* for the master path and the subparts, if any.

Specifying Arguments as nil

Except for Boolean arguments (for which `nil` is a valid value), the value `nil` is a placeholder. For example, if you do not specify the *`n_width`* argument or specify *`n_width`* as `nil`, the default value, which is the minimum width for the layer as defined in the technology file, is used.

For MPP template definitions, `nil` means the following, depending on the type of argument:

- For Boolean arguments, `nil` is a valid value, so you cannot specify `nil` as a placeholder. The following are Boolean arguments:

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

- For non-Boolean arguments, specifying `nil` causes the default value, as it is defined for the argument in the `rodCreatePath` section in *Virtuoso Relative Object Design SKILL Reference*, to be used. Specifying `nil` is equivalent to not specifying a value for the argument.

Specifying Template Arguments as Expressions

You can specify template arguments as expressions; the expressions are stored as part of the technology file. An expression must evaluate to a data type defined for the argument.

For example, you could specify the *`n_width`* argument by using the following expression:

```
2 * minWidth
```

If `minWidth` for the layer is defined as `1.5`, the *`n_width`* argument is evaluated as `3.0`. If `minWidth` is now updated to `1.2`, *`n_width`* will be evaluated as `2.4` when you next access the template.

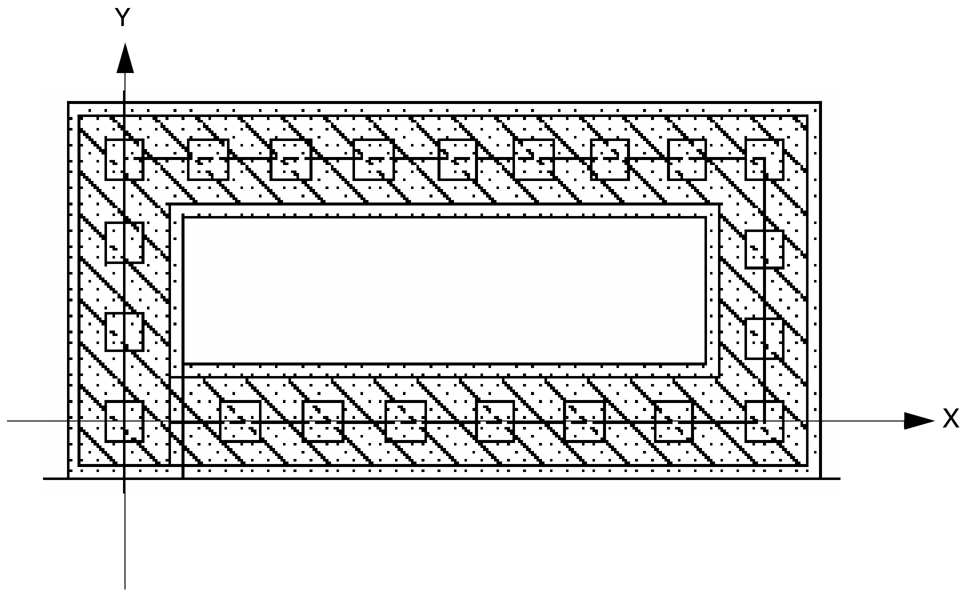
Example of an MPP Template Definition

The sample MPP template definition available at the end of this section is for a guard ring. Assuming that this template is already defined in your technology file, you can create the guard ring in a layout window by performing the following steps:

1. Choose *Create – Multipart Path*.
2. From the *MPP Template* list, choose `guardring`.
3. Click the following five points in the canvas:

   ```
   0:0  0:5  10:5  10:0.9  0.9:0.9
   ```

A guard ring, as shown in the figure below, is created:



Sample MPP template definition for the guard ring shown in the figure above:

```
;********************************
; DEVICES
;********************************
devices(
multipartPathTemplates(
; ( name [masterPath] [offsetSubpaths] [encSubPaths] [subRects] )
;
;   masterPath:
;   (layer [width] [choppable] [endType] [beginExt] [endExt] [justify] [offset]
;   [connectivity])
;
;   offsetSubpaths:
;  (layer [width] [choppable] [separation] [justification] [begOffset] [endOffset]
;   [connectivity])
;
;   encSubPaths:
;   (layer [enclosure] [choppable] [begOffset] [endOffset] [connectivity])
;
;   subRects:
;   (layer [width] [length] [choppable] [separation] [justification] [space]
    [begOffset] [endOffset] [gap] [connectivity] [beginSegOffset] [endSegOffset])
;
;   connectivity:
;   ([I/O type] [pin] [accDir] [dispPinName] [height] [ layer]
;    [layer] [justification] [font] [textOptions] [orientation]
;    [refHandle] [offset])
;
( guardring
    ; master path:
    (
        ("diff" "drawing")  ; layer-purpose pair
        1.8                 ; width
        nil                 ; choppable
    )
    ; offset subpath (there is none)
    nil
    ; enclosure subpath:
    (
        ("metal1" "drawing"); layer-purpose pair
        0.2                 ; enclosure
        t                   ; choppable
        -0.2                ; begin offset
        0.2                 ; end offset
    )
    ; set of subrectangles:
    (
        ("cont" "drawing"   ; layer-purpose pair
        nil                 ; width
        nil                 ; length
        t                   ; choppable
        nil                 ; separation
        nil                 ; justification
        nil                 ; space
        -0.6                ; begin offset
    )
  ) ; end guardring template
 ) ;multipartPathTemplates
) ;devices
```

waveguideTemplates

```
waveguideTemplates(
     ( t_name ( tx_masterLayer tx_masterPurpose )
          ['minWidth n_minWidth]
          ['minBendRadius n_minBendRadius]
          ['modeProperties l_modeProperties]
          ['maxTaperAngle f_maxTaperAngle]
          'derivedShapeSpecs(( tx_layer | ( tx_layer tx_purpose )
          {    'type "enclosure" ['enclosure tn_enclosure] |
               'type "offset" ['side "both" | "right" | "left"]
                    ['inner tn_inner] ['outer tn_outer]
          }
          )...
          );derivedShapeSpecs
     );t_name
 );waveguideTemplates
```

Description

(ICADVM20.1 Only – Photonics) Defines a template or a series of templates that specify waveguide objects. In Photonic Integrated Circuits, a waveguide is a 3D structure built as a vertical stack of shapes on neighboring layers. One of the layers forms the master, and the shapes on the other layers are relative to the master with some enclosure or offset.

For information about creating waveguides by using the graphical user interface, see *Editing Composite Waveguides* in Virtuoso Layout Suite documentation.

Arguments

t_name	Name of the template. The name must be unique within the `waveguideTemplates` section. Do not assign the name `New`; it is a reserved name.

tx_masterLayer tx_masterPurpose

	The layer-purpose pair (LPP) on which the `waveguideDef` object is to be created.
	
	You can specify the LPP as a list of strings enclosed in double quotes and separated by a space:
	
	`list("`*`t_layerName`*`" "`*`t_purposeName`*`")`
	
	Alternatively, you can specify a list of integers separated by a space:
	
	`list(`*`x_layerNumber x_purposeNumber`*`)`

`n_minWidth`	The minimum width of the object.
`n_minBendRadius`	The minimum bend radius of the object.

`l_modeProperties`	The mode properties of the waveguide object as a list in the following format:
	
	`(`
	`'function `*`t_function`*
	`'names ( (`*`t_modeName n_modeIndex`*`)...  )`
	`)`
	
	Here:
	
	■ *`t_function`*: The name of a SKILL function that returns the mode properties of the waveguide.
	
	■ *`t_modeName n_modeIndex`*: One of a series of tuples representing mode properties. Each tuple must be unique in the series and must include a mode name (a string) and an index (an integer greater than zero).

`f_maxTaperAngle`	The maximum angle that can be used when tapering a wave guide. A value of `0` indicates that tapering is not allowed. A value of `90` indicates that tapering can be done at a straight angle.
`l_derivedShapeSpecs`	

The object shape to be derived from the master shape. A derived shape definition list has the following format:

```
list((tx_layer | (tx_layer tx_purpose)
{['type "enclosure" tn_enclosure] | ['type
"offset" ['side "both" | "right" | "left"]
['inner tn_inner] ['outer tn_outer]]}…)
```

Here:

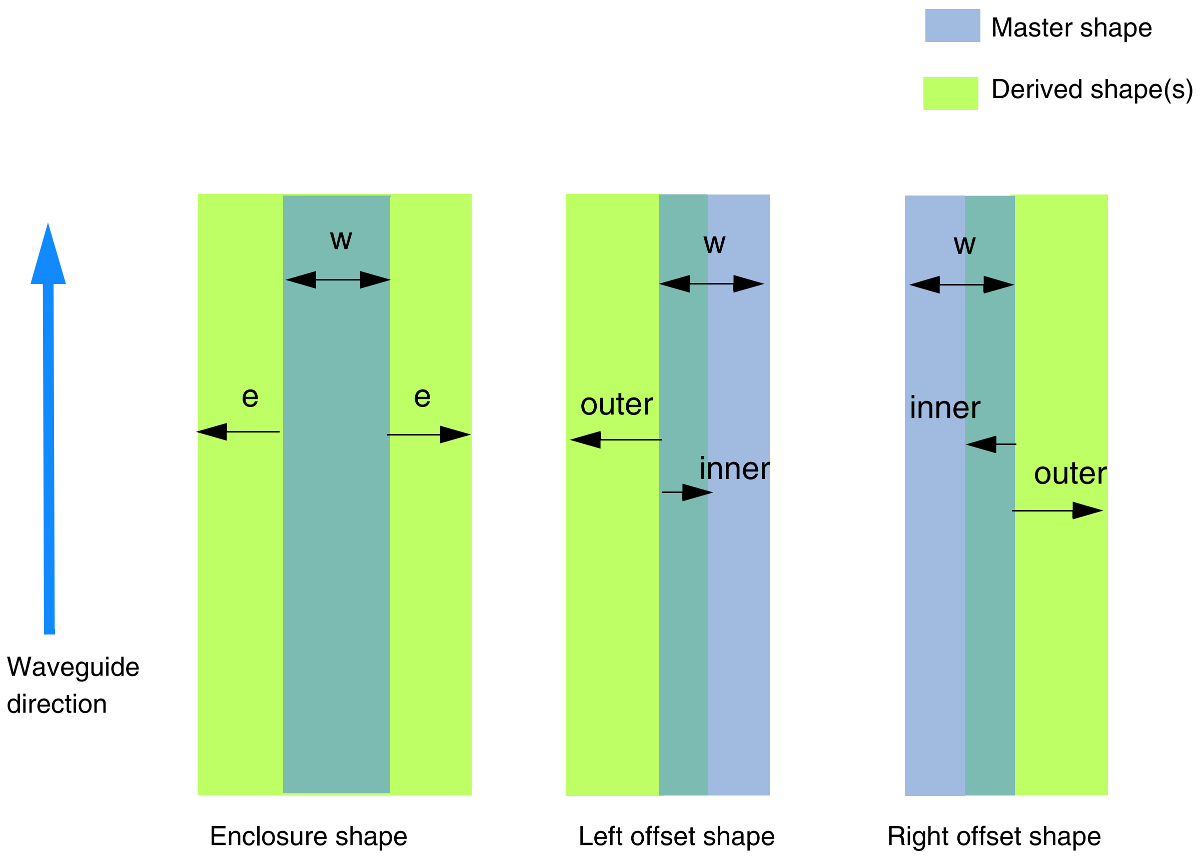
- *`tx_layer`*: The layer of the derived shape.
- *`tx_purpose`*: The corresponding purpose for the layer. If not specified, the value is set as `drawing`.
- `type`: The type of the derived shape, which can be either of the following:
  - ❑ `enclosure`: Centers the shape around the master shape based on the *`tn_enclosure`* value specified. By default, the value is `nil`, which implies zero.
  - ❑ `offset`: Offsets the shape from the master shape. The specification can include none or some of these attributes: `side` (direction as `both`, `right`, or `left`), `inner` (offset of the inner edge of the derived shape from the centerline of the master shape), and `outer` (offset of the outer edge of the derived shape from the edge of the master shape). The default values are `both`, `nil` (which implies w/2), and `nil` (which implies zero).

The *`tn_enclosure`*, *`tn_inner`*, and *`tn_outer`* values can be specified either as a string (in which case, they are interpreted and stored, without validation, as SKILL expressions) or as fixed numbers.

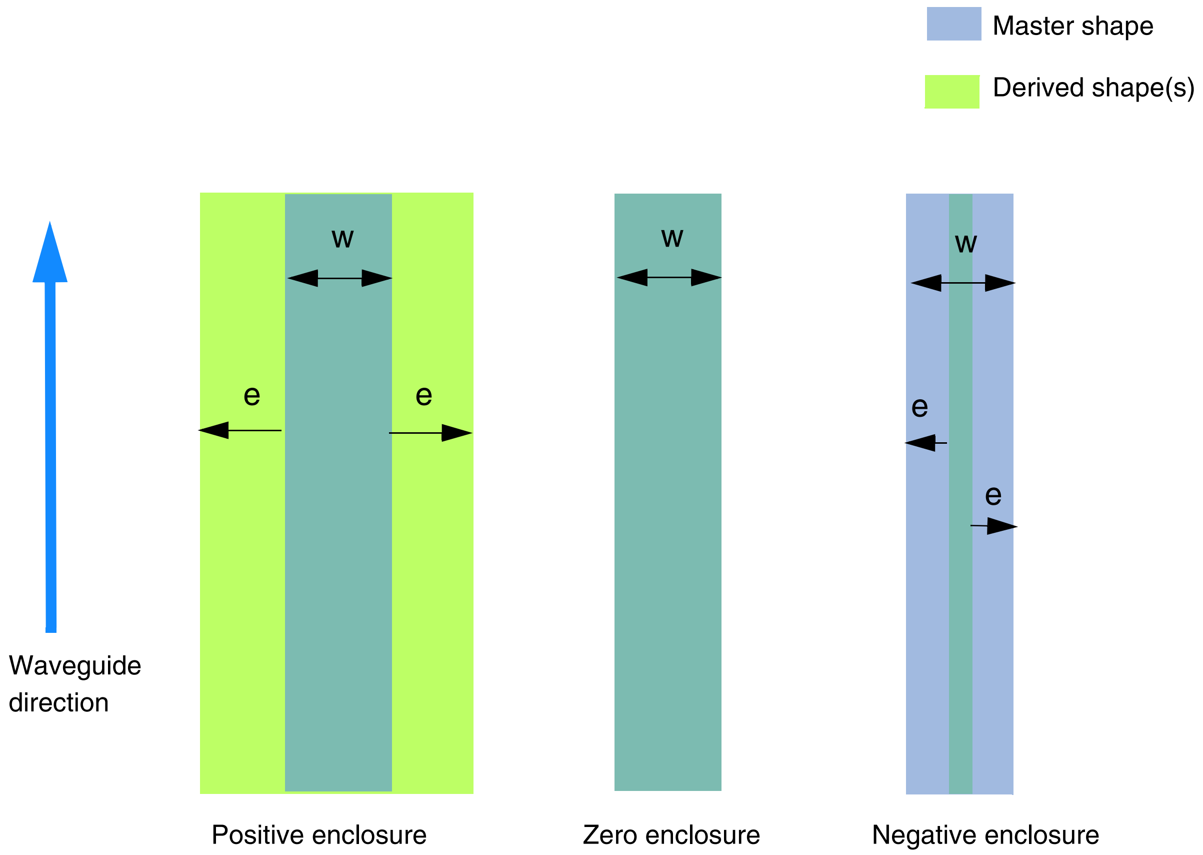
For more information about this argument, see “Deriving Shapes”.

Deriving Shapes

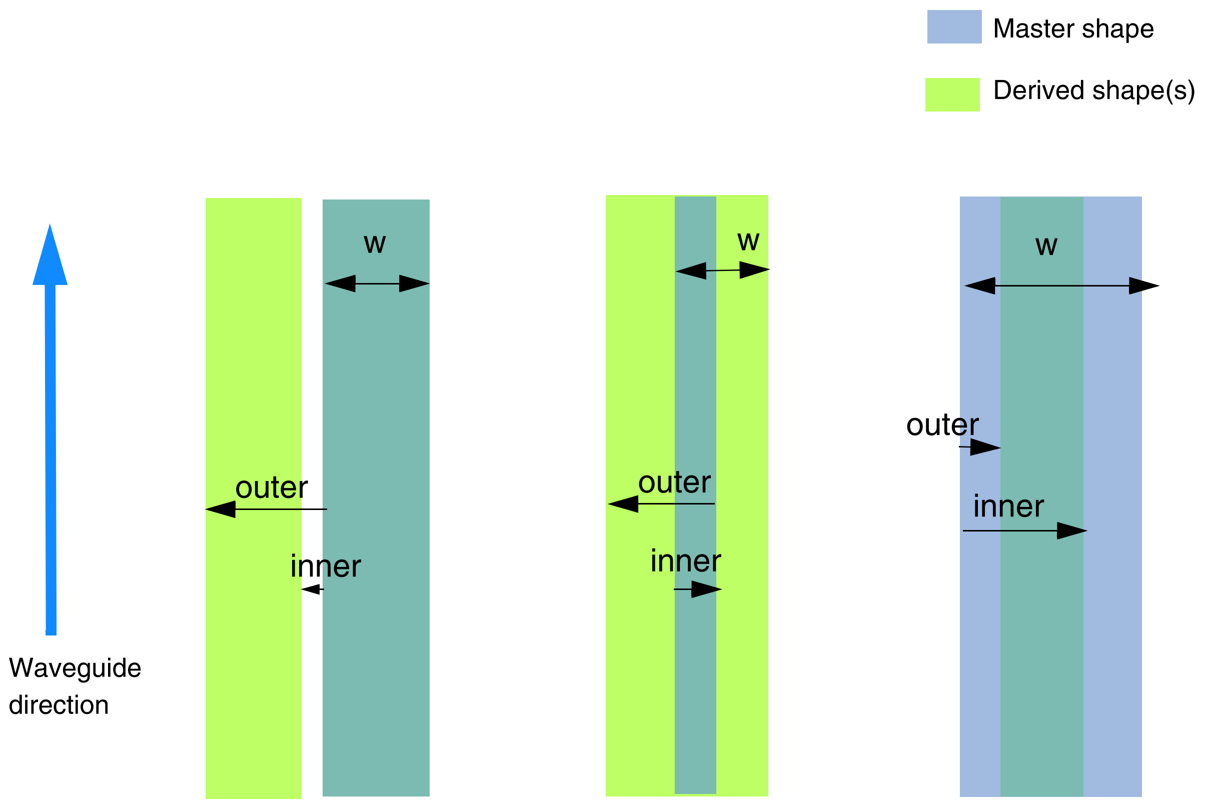
The following figure illustrates the three kinds of derived shapes.



The following figure illustrates the direction in which the enclosure for enclosure shapes is applied. A positive number extends the enclosure outside the master shape. A zero enclosure results in a derived shape with the same dimensions as the master shape. A negative number makes the derived shape skinnier than the master shape.



The following figures illustrates the direction in which the enclosure for offset shapes is applied. A positive number extends away from the edge of the master shape. A negative number is directed from the edge of the master shape into the master shape.



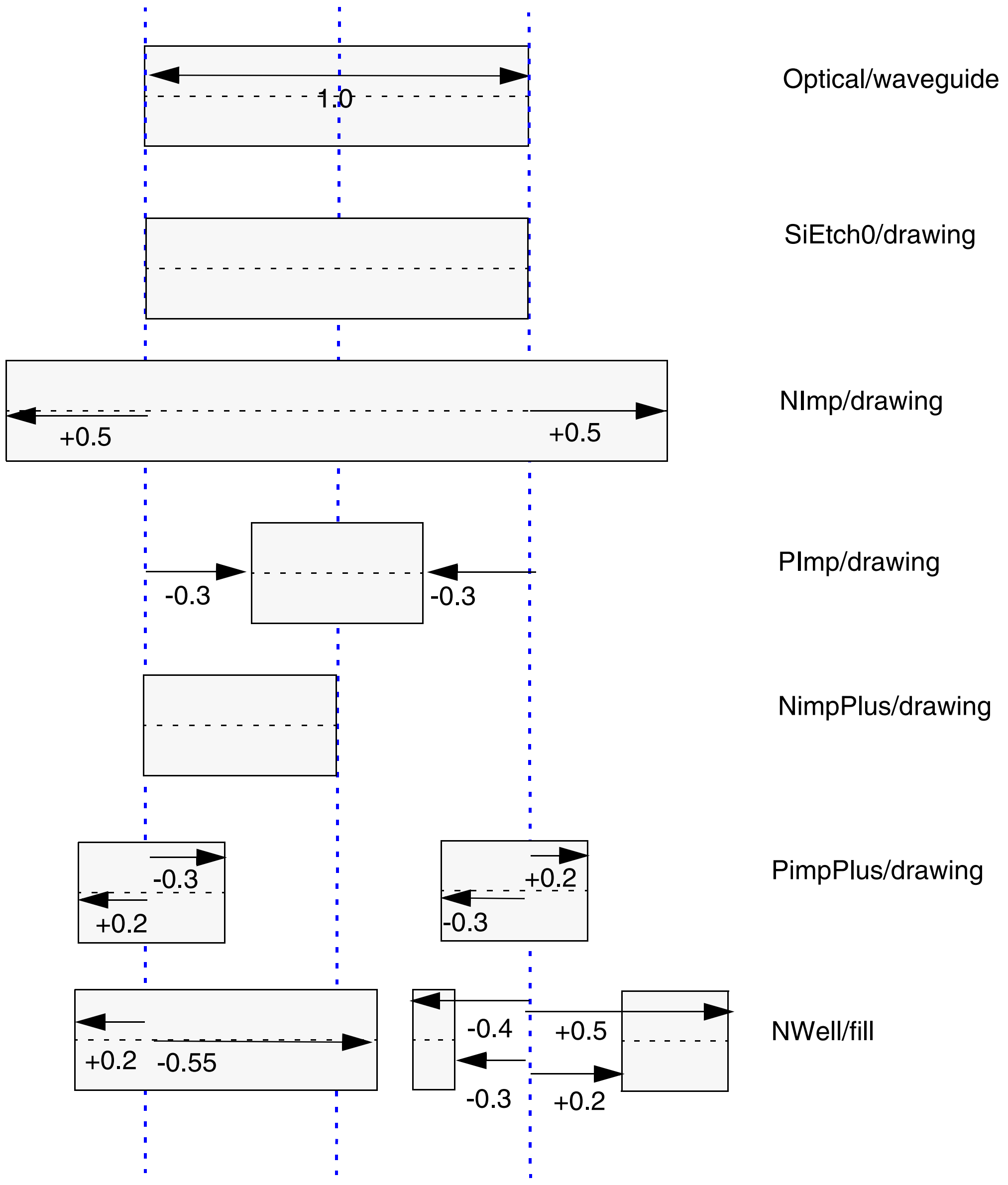
Positive enclosures Positive Outer; Negative Inner Negative enclosures

Example of a Waveguide Template Definition

Let us consider the following sample waveguideTemplate definition.

```
waveguideTemplates(
    ( "simpleWaveguide" ( "Optical" "waveguide" )
        'minWidth 0.5
        'minBendRadius 13.2
        'modeProperties ('function  "myFunc"  'names (( "mode1" 1) ("mode2" 2)))
        'derivedShapeSpecs (
        (("SiEtch0" "drawing") 'type "enclosure" 'enclosure 0.8)
        (("NImp" "drawing") 'type "enclosure" 'enclosure "0.5*w")
        (("PImp" "drawing") 'type "enclosure" 'enclosure -0.3 )
        (("NimpPlus" "drawing") 'type "offset" 'side "left")
        (("PimpPlus" "drawing") 'type "offset" 'inner -0.3 'outer  0.2)
        (("NWell" "fill") 'type "offset" 'side "right" 'inner "-0.4*w"
            'outer "-0.4*w + 0.1")
        (("NWell" "fill") 'type "offset" 'side "left" 'inner -0.55 'outer 0.2)
        (("NWell" "fill") 'type "offset" 'side "right" 'inner  0.2 'outer 0.5)
    ) ;simpleWaveguide
) ;waveguideTemplates
```

The following figure illustrates one example of such a waveguide. It shows a cross section, and the waveguide direction is assumed to be into the page, in the direction of the waveguide. As a result, the left side for a derived shape with the type `offset` is to the left of the master path. In this case, the master path width is `1.0` and all the expressions are calculated with `w=1.0`.



extractMOS

```
extractMOS(
     t_deviceName
     tx_recognitionLayer
     tx_gateLayer
     tx_sourceDrainLayer
     tx_bulkLayer
     [ tx_modelName ]
) ;extractMOS
```

Description

Defines the layers used to recognize and establish connections to a MOS transistor.

Arguments

tx_deviceName	The name of the device.
tx_recognitonLayer	The MOS recognition layer. Usually a derived layer. Valid values: Layer name or layer number
tx_gateLayer	The gate layer. Valid values: Layer name or layer number of a connected type layer
tx_sourceDraineLayer	The source-drain layer. Valid values: Layer name or layer number of a connected type layer
tx_bulkLayer	The bulk layer. Valid values: Layer name or layer number of a connected type layer; can be `substrate`
tx_modelName	The spice model name; used when the MOS transistor name is different from the model name.

Example

```
devices(
    extractMOS( "NMOS"  ngate  poly  oxide  substrate "lvn" )
    extractMOS( "PMOS"  pgate  poly  oxide  nwell     "lvp" )
)
```

extractRES

```
extractRES(
     tx_deviceName
     tx_recognitionLayer
     tx_termLayer
     [ tx_modelName ]
) ;extractRES
```

Description

Defines the layers used to recognize and establish connections to a resistor.

Arguments

tx_deviceName	The name of the device.
tx_recognitonLayer	The resistor recognition layer. Usually a derived layer. Valid values: Layer name or layer number
tx_termLayer	The terminal connection layer. Valid values: Layer name or layer number of a connected type layer
tx_modelName	The spice model name; used when the resistor name is different from the model name.

Example

```
devices(
    extractRES( "polyR"  polyRes  poly  rp )
)
```

extractCAP

```
extractCAP(
     t_deviceName
     tx_recognitionLayer
     tx_plusLayer
     tx_minusLayer
     [ tx_modelName ]
) ;extractCAP
```

Description

Defines the layers used to recognize and establish connections to a capacitor.

Arguments

tx_deviceName	The name of the device.
tx_recognitonLayer	The capacitor recognition layer. Usually a derived layer. Valid values: Layer name or layer number
tx_plusLayer	The plus terminal layer. Valid values: Layer name or layer number of a connected type layer
tx_minusLayer	The minus terminal layer. Valid values: Layer name or layer number of a connected type layer
tx_modelName	The spice model name; used when the capacitor name is different from the model name.

Example

```
devices(
    extractCAP( "polyC"  polyCap  poly1  poly2  cp )
)
```

extractDIODE

```
extractDIODE(
     t_deviceName
     tx_recognitionLayer
     tx_plusLayer
     tx_minusLayer
     [ tx_modelName ]
     )
```

Description

Defines the layers used to recognize and establish connections to a diode.

Arguments

tx_deviceName	The name of the device.
tx_recognitonLayer	The diode recognition layer. Valid values: Layer name or layer number
tx_plusLayer	The plus terminal layer. Valid values: Layer name or layer number of a connected type layer; cannot be the same layer as the minus terminal layer.
tx_minusLayer	The minus terminal layer. Valid values: Layer name or layer number of a connected type layer; cannot be the same layer as the plus terminal layer
tx_modelName	The spice model name; used when the diode name is different from the model name.

Example

```
devices(
    extractDIODE( "nsubDiode"  ndiode  ndiff  substrate )
)
```

9

Technology File LSW Layer Specifications

This chapter contains the following topics pertaining to LSW Layer specifications:

leRules

```
leRules()
```

Description

Is the technology file enclosure for specifying LSW layers.

leLswLayers

```
leLswLayers(
     ( t_layerName t_layerPurpose )
     …
) ;leLswLayers
```

Description

Lists layer-purpose pairs in the order in which they must appear in the Layers panel of the Palette. If you do not define this subsection in the technology file, the layer-purpose pairs are displayed in the Palette based on how they are defined in the techLayerPurposePriorities subsection of the `layerDefinitions` section.

The `leLswLayers` subsection must be contained within the `leRules` section enclosure.

Note: The `leLswLayers` subsection specifies the default list that appears in the Palette. The Palette provides commands that let you modify while you work the layer-purpose pairs and the order in which they appear.

Arguments

`t_layerName`	The name of the layer.
`t_layerPurpose`	The purpose of the layer.

Example

```
leLswLayers(
 ; ( layer     purpose )
   ( metal1    drawing )
   ( metal2    drawing )
   ( metal3    drawing )
   ( poly1     drawing )
   ( pwell     drawing )
   ( nimplant  drawing )
   ( diff      drawing )
)
```

Defines the layer-purpose pairs and the order in which they are displayed in the Palette.

10

Technology File Packaging Definitions

This chapter contains the following topics pertaining to packaging definitions:

- packaging
- fingerDefinitions
- wireProfiles
- wireProfileGroups

packaging

```
packaging()
```

Description

(ICADVM20.1 Only – Virtuoso MultiTech Framework) Specifies packaging definitions. Subsections must be enclosed within the parentheses of this section.

fingerDefinitions

```
fingerDefinitions(
     ( t_name
          l_typeSpec
     )
     …
) ;fingerDefinitions
```

Description

(ICADVM20.1 Only – Virtuoso MultiTech Framework) Specifies a finger definition.

Arguments

t_name	The name of the finger definition.

l_typeSpec

A list indicating the type-specific details. The list format varies depending on the type of the finger definition, which can be one of the following:

- `oblong`: A list specifying the length, width, and layer of an oblong finger definition in the following syntax:

```
('type "oblong"
   'length f_length
   'width f_width
   'layer tx_layer
 )
```

- `retangle`: A list specifying the length, width, and layer of a rectangular finger definition in the following syntax:

```
('type "rectangle"
   'length f_length
   'width f_width
   'layer tx_layer
 )
```

- `circle`: A list specifying the diameter and layer of a circular finger definition in the following syntax:

```
('type "circle"
   'diameter f_diameter
   'layer tx_layer
 )
```

- `padStack`: A list specifying the lib/cell/view for a padstack finger definition in the following syntax:

```
('type "padStack"
   'libName t_libName
   'cellName t_cellName
   'viewName t_viewName
 )
```

Example

```
packaging(
    fingerDefinitions(
        ( "myFingerDef1"
            ('type "oblong"
                'length 0.5
                'width 0.3
                'layer "metal1"
            )
            ('type "rectangle"
                'length 0.5
                'width 0.3
                'layer "metal1"
            )
            ('type "circle"
                'diameter 0.5
                'layer "metal1"
            )
            ('type "oblong"
                'libName "lib1"
                'cellName "cell2"
                'viewName "view3"
        );myFingerDef1
    );fingerDefinitions
);packaging
```

Related Topics

- packaging

wireProfiles

```
wireProfiles(
     ( t_name
       'direction   "forward" | "reverse"
       ['diameter   f_diameter]
       ['material   t_material]
       ['fromVendor t | nil]
       ['color      n_color]
       ['visibility t | nil]
       'points      l_points
     )
     …
) ;wireProfiles
```

Description

(ICADVM20.1 Only – Virtuoso MultiTech Framework) Specifies wire profile definitions.

Arguments

`t_name`	The name of the wire profile.
`'direction`	The direction of the wire profile. Valid values: `forward`, `reverse`
`'diameter` *`f_diameter`*	The diameter of the wire profile. Default value: `0`
`'material` *`t_material`*	The wire profile material name as a string. Default value: an empty string
`'fromVendor`	Indicates whether the wire profile is from a vendor. Boolean value: `nil` (default), `t`
`'color` *`n_color`*	The color index of the wire profile. Default value: `0`
`'visibility`	The visibility of the wire profile. Boolean value: `nil` (default), `t`
`'points` *`l_points`*	A list of points of the type `switch` or `point`.

The list has the following syntax:

```
(
    { l_hvPoint | l_switchPoint } …
)
```

These arguments are described as follows.

- *`l_hvPoint`*: A list with the following syntax:

```
(
   'type "point"
      'horizontal l_point
      'vertical l_point
   ['horizontalTurn f_hTurn]
)
```

Here:

- The `type` value is `point`.
- A horizontal component and a vertical component are specified as a list followed by the argument `horizontal` and `vertical`, respectively. It is specified as a list with the following syntax:

  ```
  (
      'type t_type
      'value f_value
      ['locked b_locked]
      ['max f_max]
      ['min f_min]
      ['step f_step]
   )
  ```

  Here:

  - *`t_type`*: Point type for the component. Valid values are `length`, `percent`, `angle`.
  - *`f_value`*: Value corresponding to *`t_type`*. For length, it is in microns. For percent, it is a value between `0` and `100`. For angle, it is a value between `-90` and `+180`.
  - *`b_locked`*: Boolean value indicating whether the point is locked by the manufacturer. Values: `nil` (default) or `t`.
  - *`f_max`*: Maximum length, when *`t_type`* is `length`.
  - *`f_min`*: Minimum length, when *`t_type`* is `length`.
  - *`f_step`*: The step when *`t_type`* is `length`. The default value is `0`.
- *`f_hTurn`*: Specifies a horizontal angle in the wire. Valid values: a float number in the range `-90` to `+90`. Default value: `0`

- *`l_switchPoint`*: Indicates a switch of direction. If the profile has *`t_direction`* set to `forward`, then all subsequent points are treated as being in the `reverse` direction. The `type` value is `switch`. It is a list with the following syntax:

  ```
  ('type "switch")
  ```

Example

```
packaging(
    wireProfiles(
        ( "myProfile1"
            'direction      "forward"
            'diameter       20.0
            'material       "Gold"
            'fromVendor     t
            'color          0
            'visibility     t
            'points         (
                            'type "point"
                                'horizontal (
                                            'type "length"
                                            'value 100.0
                                            'locked t
                                            )
                                'vertical   (
                                            'type "percent"
                                            'value 50.0
                                            )
                            )
                            (
                            'type "switch"
                            )
                            (
                            'type "point"
                                'horizontal (
                                            'type "length"
                                            'value 50
                                            )
                                'vertical   (
                                            'type "angle"
                                            'value -25.0
                                            'locked nil
                                            )
                            'horizontalTurn 30.0
                            )

                );points
        );myProfile1
    );wireProfiles
);packaging
```

Related Topics

- packaging

wireProfileGroups

```
wireProfileGroups(
     ( t_name
       ['vendor t_vendorName]
       ['logoFile   t_logo]
       'wireProfiles l_wireProfileNames
     )
     …
) ;wireProfileGroups
```

Description

(ICADVM20.1 Only – Virtuoso MultiTech Framework) Specifies a wire profile group.

Arguments

t_name	The name of the wire profile group.
'vendor *t_vendorName*	The name of the vendor as a string. Default value: an empty string
'logoFile *t_logo*	The logo filename as a string. Default value: an empty string
'wireProfiles *l_wireProfileNames*	A list of wire profile names to be included in the group.

Example

```
packaging(
    wireProfileGroups(
        ( "KS_group"
            'vendor     "Kulicke and Soffa"
            'logoFile   "wbProfileKNSLogo.bmp"
            'wireProfiles ("myProfile1" "myProfile2")
    );wireProfileGroups
);packaging
```

Related Topics

- packaging

11 Display Resource File

The display resource file defines the packets that the Cadence® design software use to display the layers in your design. It also defines the different display devices, such as monitors and plotters, that you use.

This chapter described the sections in the `display.drf` file in the following topics:

drDefineDisplay

```
drDefineDisplay(
     (t_displayName)
     …

)
```

Description

Lists the display devices for which display resources are defined in the display resource file.

The following table lists commonly used display devices.

Device Name	Type
display	Color monitor
hp6	Hewlett-Packard 6-carousel pen plotters
hp8	Hewlett-Packard 8-carousel pen plotters
psb	PostScript black-and-white plotters
versatecb	Versatec and CalComp black-and-white plotters
versatecc	Versatec and CalComp color plotters
XBlackWhite	Black-and-white X Window System monitors
X4PlaneColor	4-plane color X Window System monitors

For more information about setting up plotters, see *Plotter Configuration User Guide*.

Arguments

t_displayName	The name of the display device. Valid values: Any unique string

Example

```
drDefineDisplay(
;( DisplayName )
 ( display )
 ( psb )
)
```

Defines display devices `display` and `psb`.

drDefineColor

```
drDefineColor(
     (t_displayName t_colorName x_red x_green x_blue [g_blink])
     …
)
```

Description

Defines the colors for a display device.

Arguments

t_displayName	The name of the display device. Valid values: A display device listed in `drDefineDisplay`
t_colorName	The name of the color. Valid values: Any unique string
x_red	The red index for the color. Valid values: An integer from `0` through `255`, inclusive
x_green	The green index for the color. Valid values: An integer from `0` through `255`, inclusive
x_blue	The blue index for the color. Valid values: An integer from `0` through `255`, inclusive
g_blink	Indicates that the color is blinking. Valid values: `t`, `nil` Default: `nil`

Note: When you define a blinking and non-blinking version of the same color, we recommend that you append the character `B` to the name of the non-blinking color to name the blinking color, such as `white` and `whiteB`.

Example

```
drDefineColor(
;( DisplayName    ColorName    Red    Green    Blue   Blink )
 ( display        white        255    255      255          )
 ( display        whiteB       255    255      255    t     )
 ( display        yellow       255    255      0            )
 ( display        silver       217    230      255          )
 ( display        cream        255    255      204          )
 ( display        pink         255    191      242          )
 ( display        magenta      255    0        255          )
 ( display        lime         0      255      0            )
 ( display        tan          255    230      191          )
 ( display        cyan         0      255      255          )
 ( display        cadetBlue    57     191      25           )
 ( display        orange       255    128      0            )
 ( display        red          255    0        0            )
 ( display        purple       153    0        230          )
 ( display        green        0      204      102          )
 ( display        brown        191    64       38           )
 ( display        blue         0      0        255          )
 ( display        blueB        0      0        255    t     )
)
```

Defines colors for the display device `display`.

drDefineStipple

```
drDefineStipple(
     (t_displayName t_stippleName l_stipplePattern)
     …
)
```

Description

Defines the stipple patterns for the display devices.

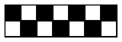
Arguments

t_displayName	The name of the display device. Valid values: A display device defined in `drDefineDisplay`
t_stippleName	The name of the stipple pattern. Valid values: Any unique string
l_stipplePattern	A list of lists indicating the pattern of the stipple. The list has the following syntax: `( l_pattern … )` where, *l_pattern* is a pattern list defining the pattern of a portion of the stipple. Use `1` or `t` to indicate where the pattern is solid and `0` or `nil` to indicate where the pattern is blank.

Sample Stipple Patterns

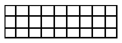
The following illustrate a checker stipple pattern and a grid stipple pattern:

checker



```
((1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
 (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
 (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
 (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
 (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
 (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
 (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
 (1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0)
 (0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)
 (0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)
 (0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)
 (0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)
 (0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)
 (0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)
 (0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)
 (0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1)
)
```

grid



```
((0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1)
 (0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1)
 (0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1)
 (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
 (0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1)
 (0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1)
 (0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1)
 (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
 (0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1)
 (0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1)
 (0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1)
 (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
 (0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1)
 (0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1)
 (0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1)
 (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
)
```

Example

```
drDefineStipple(
;( DisplayName    StippleName
;  StipplePattern )
 ( display        solid
  (
   (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
   (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
   (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
   (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
   (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
   (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
   (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
   (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
   (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
   (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
   (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
   (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
   (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
   (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
   (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
   (1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1)
  )
 )
 ( display        dots
  (
   (0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0)
   (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
   (0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1)
   (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
   (0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0)
   (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
   (0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1)
   (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
   (0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0)
   (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
   (0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1)
  )
 )
) ;drDefineStipple
```

Defines a solid stipple pattern and a stipple pattern with dots for the display device `display`.

drDefineLineStyle

```
drDefineLineStyle(
     (t_displayName t_lineStyle x_size l_pattern)
     …
)
```

Description

Defines the line styles for the display devices.

Arguments

t_displayName	The name of the display device. Valid values: A display device defined in `drDefineDisplay`
t_lineStyleName	The line style name. Valid values: Any string
x_size	The thickness of the line pattern, specified in pixels.
l_pattern	A list of binary values indicating the pattern of the line. Specify `1` or `t` to indicate where the line is solid and `0` or `nil` to indicate where the line breaks.

Sample Line Styles

The following illustrate a dashed line style, a line style made of dots, and a line style that is a combination of dashes and dots:

dashed - — — — — — — -	`list(1 1 1 1 0 0)`
dots - - - - - - - - - - - - - - - - - -	`list(1 0 0)`
dashDot – - — - — - — - –	`list(1 1 1 0 0 1 0 0)`

Example

```
drDefineLineStyle(
;( DisplayName       LineStyle     Size      Pattern                )
 ( display           solid          1        (1 1 1)                )
 ( display           dashed         1        (1 1 1 1 0 0)          )
 ( display           dots           1        (1 0 0)                )
 ( display           dashDot        1        (1 1 1 0 0 1 0 0)      )
 ( display           shortDash      1        (1 1 0 0)              )
 ( display           doubleDash     1        (1 1 1 1 0 0 1 1 0 0) )
 ( display           hidden         1        (1 0 0 0)              )
 ( display           thickLine      3        (1 1 1)                )
) ;drDefineLineStyle
```

Defines a number of line styles for the display device `display`.

drDefinePacket

```
drDefinePacket(
     (t_displayName t_packetName t_stippleName
      t_lineStyleName t_fillColor t_outlineColor
      [t_fillStyle]
     )
     …
)
```

Description

Defines packets of display information. Packets are used in the technology file to define how layers look.

Arguments

t_displayName	The name of the display device. Valid values: A display device defined in `drDefineDisplay`
t_packetName	The name of the display packet. Valid values: Any string; the packet name must be unique for a display device See Guidelines for Naming Display Packets for more information.
t_stippleName	The name of the stipple pattern. Valid values: A stipple defined in `drDefineStipple`
t_lineStyleName	The name of the line style. Valid values: A line style defined in `drDefineLineStyle`
t_fillColor	The name of the fill color. Valid values: A fill color defined in `drDefineColor`
t_outlineColor	The name of the outline color. Valid values: An outline color defined in `drDefineColor`
t_fillStyle	The name of the fill style. If specified, this value overrides stipple and line styles. Valid values: `outline`, `solid`, `x`, `stipple`, `outline stipple` See Fill Styles for more information.

Guidelines for Naming Display Packets

We recommend that you adhere to the guidelines outlined in this section while naming a display packet.

A display packet name must have the following structure:

`color[stipple][line][_[S][L][N][B]]`

where,

`color`

Specifies the fill and outline colors in the following format:

`[ fill ] [ outline ]`

At least one color name must be specified.

Specify colors according to the following rules:

- When the fill and outline colors are different, specify the fill color followed by the outline color.
- When the fill and outline colors are the same, specify that one color.
- When the fill and outline colors are the same, but the outline color is blinking, append `B` to the color name when specifying the outline color; for example, `white` and `whiteB`. If the outline color name is constructed in any other way (for example, `white2`), specify both colors (`whitewhite2`).

`stipple`

Specifies a stipple pattern name. This field is optional; if a value is not specified, it indicates the default stipple pattern, `blank`.

`line`

Specifies a line style name.

- Default (when the packet name does not contain `N` in the extension): `solid`
- Default (when the packet name contains `N` in the extension: `none`

`_SLNB` extension	■ `S` indicates that the stipple pattern name is specified and the line style is the default (`solid`). ■ `SN` indicates that the stipple pattern name is specified and the line style specified is `none`. ■ `L` indicates that the line style name is specified and the stipple pattern is the default (`blank`). ■ `N` indicates that the line style specified is `none`. ■ `B` indicates that the outline color is a blinking color. Only the outline color can be a blinking color; the fill color must be non-blinking. See drDefineColor for information about defining colors.

Important

The packet name must not contain spaces and the underscore character (_) must precede the `SLNB` extension.

The following table shows sample combinations of color, stipple, and line resources and the packet names correctly built according to the packet naming convention.

Fill Color	**Outline Color**	**Stipple Pattern**	**Line Style**	**Packet Name**
blue	blue	blank	solid	`blue`
blue	blue	blank	dashed	`bluedashed_L`
blue	blue	solid	solid	`bluesolid_S`
blue	blue	metal1S	solid	`bluemetal1S_S`
blue	blue	metal1S	none	`bluemetal1S_SN`
cream	white	contp	solid	`creamwhitecontp_S`
green	green	brick	mLine	`greenbrickmLine`
red	red	X	thickLine	`redXthickLine`
red	Blinking red named `redB`	x	solid	`redx_SB`
red	Blinking red named `red2`	solid	none	`redred2solid_SNB`

Fill Styles

The table below defines the fill styles.

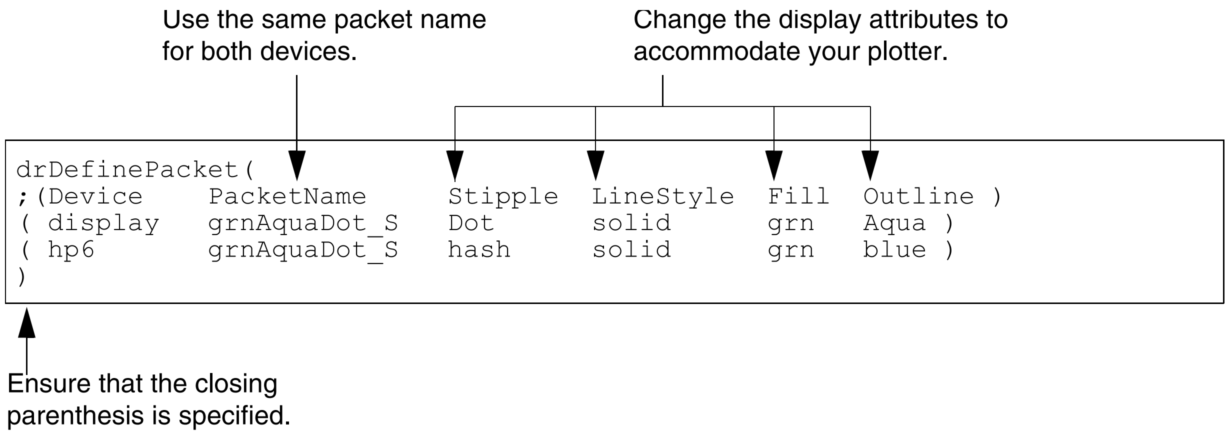
Note: A fill style overrides stipple and line styles.

Fill Style	Result	
`outline`	□	
`solid`	■	
`X`	☒	
`stipple`	▥	(The specified stipple pattern determines the fill.)
`outline stipple`	▥	(The specified stipple pattern determines the fill.)

Customizing Display Packets for Plotting

You can set up your display resource file so that a display packet appears differently on different display devices. For example, if your plotter uses only seven colors and you display your design using 12 colors, you can modify the display packets used by the layers in your design specifically for the plotting device.

You can also define a display packet with the same name, but with different content for each display device, as shown in the following example:


Use the same packet name for both devices.
Change the display attributes to accommodate your plotter.
Ensure that the closing parenthesis is specified.

```
drDefinePacket(
;(Device    PacketName     Stipple  LineStyle  Fill  Outline )
( display   grnAquaDot_S   Dot      solid      grn   Aqua )
( hp6       grnAquaDot_S   hash     solid      grn   blue )
)
```

Example

```
drDefinePacket(
;(Display  PacketName           Stipple  LineStyl  Fill    Outline   fillSt)
 ( psb     red                  blank    solid     red     red       X     )
 ( psb     red_B                blank    solid     red     redB            )
 ( psb     redredblnk_B         blank    solid     red     redblnk   solid )
 ( psb     white                blank    solid     red     white           )
 ( psb     whitedots_L          blank    dots      red     white           )
 ( psb     redhash_SN           hash     none      red     white           )
 ( psb     blueyellowslash_S    slash    solid     blue    yellow          )
 ( psb     blueyellowslash_SN   slash    none      blue    yellow    solid )
 ( psb     whitered_N           blank    none      white   red             )
 ( psb     greenblackBdot1_SB   dot1     solid     green   blackB          )
)
```

Defines display packets for the display device `psb`.

drDefinePacketAlias

```
drDefinePacketAlias(
     (t_displayName t_packetAlias t_packetName)
     …

)
```

Description

Defines an alias for the specified packet. As a result, you can use the display packet name as well as the alias to access the display packet. This allows flexibility in assigning display packets to display devices. You can alias an existing display packet name to another display packet name to change the display packet in use for a given display device.

Arguments

t_displayName	The name of the display device. Valid values: A display device defined in `drDefineDisplay`
t_packetAlias	The alias for the display packet name. Valid values: A display device defined in `drDefineStipple`
t_packetName	The name of the display packet for which the alias needs to be defined. Valid values: A packet defined in `drDefinePacket`

Example

```
drDefinePacketAlias(
    ("display" "redSolid1" "redSolid")
)
```

Defines `redSolid1` as an alias for the display packet `redSolid` when used with the display device `display`.