Technology File and Display Resource File SKILL Reference Manual

Product Version 5.0
January 2003

Cadence Design Systems, Inc., 555 River Oaks Parkway, San Jose, CA 95134, USA

Contents

Preface

The *Technology File and Display Resource File SKILL Reference Manual* provides detailed information about the Cadence® SKILL language functions that operate on the technology files and display reference files you use with your Cadence Design Framework II (DFII) designs.

The preface discusses the following:

Related Documents

The following documents give you more information about the technology file and the products that use it.

- For information about installing the product, see the *Cadence Installation Guide*.
- For more information about defining and using technology files and display resource files, see the *Technology File and Display Resource File User Guide* and the *Technology File and Display Resource File ASCII Syntax Reference Manual*.
- For known problems and solutions for the technology file, see *Technology File Known Problems and Solutions*.
- For known problems and solutions for the display resource file, see *Display Resource Editor Known Problems and Solutions*.
- For information about new features, enhancements, and PCRs fixed in this release for the technology file, see the *Technology File Product Notes*.
- For information about new features, enhancements, and PCRs fixed in this release for the display resource file, see the *Display Resource Editor Product Notes*.
- For more information about the SKILL programming language, see the following:

  Design Framework II SKILL Functions Reference

  SKILL Development Help

SKILL Language Reference

SKILL Language User Guide

- For more information about SKILL access to other applications, see the following:

  Cadence Analog Design Environment SKILL Language Reference

  Virtuoso Schematic Composer SKILL Functions Reference

  Custom Layout SKILL Functions Reference

  Preview SKILL Functions Reference

Typographic and Syntax Conventions

The following sections explain the typographic and syntax conventions used in this document.

Syntax Conventions

This list describes the syntax conventions used in this document.

`text`	Indicates text you must type exactly as it is presented.
`z_argument`	Indicates text that you must replace with an appropriate argument. The prefix (in this case, *`z_`*) indicates the data type the argument can accept. Do not type the data type or underscore.
`[ ]`	Denotes optional arguments. When used with vertical bars, they enclose a list of choices from which you can choose one.
`{ }`	Used with vertical bars and encloses a list of choices from which you must choose one.
`|`	Separates a choice of options.
…	Indicates that you can repeat the previous argument.
=>	Precedes the values returned by a Cadence® SKILL language function.

/	Separates the possible values that can be returned by a Cadence SKILL language function.
text	Indicates names of manuals, menu commands, form buttons, and form fields.

Important

The language requires many characters not included in the preceding list. You must type these characters exactly as they are shown in the syntax.

Data Types

The Cadence® SKILL language and ASCII file syntax support several data types to identify the type of value you can assign to an argument. Data types are identified by a single letter followed by an underscore; for example, *t* is the data type in *t_viewNames*. Data types and the underscore are used as identifiers only: they are not to be typed.

The table below lists all data types supported by SKILL.

Prefix	Internal Name	Data Type
a	array	array
b	ddUserType	Boolean
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	gdm spec
h	hdbobject	hierarchical database configuration object
l	list	linked list
m	nmpIlUserType	nmpIl user type
M	cdsEvalObject	—
n	number	integer or floating-point number

Prefix	Internal Name	Data Type
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
s	symbol	symbol
S	stringSymbol	symbol or character string
t	string	character string (text)
u	function	function object, either the name of a function (symbol) or a lambda function body (list)
U	funobj	function object
v	hdbpath	—
w	wtype	window type
x	integer	integer number
y	binary	binary function
&	pointer	pointer type

ASCII and DFII SKILL Syntax Examples

The following examples show typical syntax characters used in the technology file and display resource file ASCII syntax and DFII SKILL.

Example 1

`list( g_arg1 [g_arg2] ...) => l_result`

This example illustrates the following syntax characters.

`list`	Plain type indicates words that you must enter literally.
g_arg1	Words in italics indicate arguments for which you must substitute a name or a value.

`( )`	Parentheses separate names of functions from their arguments.
`_`	An underscore separates an argument type (left) from an argument name (right).
`[ ]`	Brackets indicate that the enclosed argument is optional.
`...`	Three dots indicate that the preceding item can appear any number of times.
`=>`	A right arrow points to the description of the return value of the function.
`l_result`	Functions compute a data value known as the return value of the function.

Example 2

```
needNCells( s_cellType | st_userType x_cellCount) => t/nil
```

This example illustrates two additional syntax characters.

`|`	Vertical bars separate a choice of required options.
`/`	Slashes separate possible return values.

1

Administrative SKILL Functions

The functions in this chapter are administrative in the sense that they do not affect the technology data of the technology file. The function descriptions in this chapter are grouped by general type in the following sections:

- File Functions
- Attach Functions
- Dump, Load, and Trigger Functions
- Floating-Point Precision Function
- Controls Class Functions

The functions described in these sections let you do the following:

- Open and close a technology file in virtual memory
- Manage files opened in virtual memory. Includes
  - Purging virtual memory
  - Refreshing virtual memory with the disk version
  - Getting identifiers of open technology files
- Save the technology file opened in virtual memory to disk
- Attach a technology file to a Cadence® Design Framework II (DFII) library, cell, or cellview
- Dump open technology files to ASCII files
- Compile and load ASCII files to virtual memory
- Execute programs based on dump and load triggers
- Set and get control parameters for use in the technology data

File Functions

The functions in this section let you open, close, and manage files in virtual memory. The open functions provide three modes that control what remains on disk and what is loaded into virtual memory. The modes are

- Read — loads the contents of the binary technology file into virtual memory and does not allow any edits to the file
- Append — loads the contents of the binary technology file into virtual memory and does allow edits
- Write — deletes the contents of the binary technology file on disk and loads the empty file into virtual memory

  Write mode does allow edits. *Use this mode with extreme caution.*

techOpenTechFile

```
techOpenTechFile(
     t_libName
     t_fileName
     [t_mode]
     )
     => d_techFileID/nil
```

Description

Loads a binary technology file in virtual memory with a specified mode and returns the database identifier of the file.

Caution

If you open a technology file in write mode, the contents of the disk file are deleted. Use this function with extreme care.

Arguments

t_libName	The library into which to load the binary technology file.
t_fileName	The name of the binary technology file to open. Valid Value: `techfile.cds`
t_mode	The mode in which to open the file. Valid Values: `r` (read only), `w` (delete contents and load empty file), `a` (append or edit mode) **Note:** If you do not specify a mode, or if you specify `r` mode, the file can only be read. If you specify `w` mode, the contents of the disk file are deleted, or truncated, and an empty technology file is loaded into virtual memory. If you specify a mode, the technology file is loaded in append mode, which means that you can edit the contents of the file.

Return Values

d_techFileID	Database identifier of the technology file loaded into virtual memory.

`nil`	The technology library or file does not exist.

Example

```
techFileID = techOpenTechFile("cellTechLib" "techfile.cds" a)
=> db:25675212
```

Sets the variable `techFileID` to the database identifier for the technology file stored in the `cellTechLib` library and loads the technology file into virtual memory in append mode.

techOpenDefaultTechFile

```
techOpenDefaultTechFile(
     )
     => d_techFileID/nil
```

Description

Loads the default binary technology file in virtual memory in read mode and returns the database identifier. The default technology file resides at the following location:

your_install_dir/tools/dfII/samples/techfile/default.cds

Arguments

None.

Return Values

d_techFileID	Database identifier of the technology file loaded into virtual memory.
nil	The technology file or library does not exist or is not at the expected location.

Example

```
techFileID = techOpenDefaultTechFile()
=> db:23676263
```

Loads the default technology file from the installation directory into virtual memory. Returns the database identifier.

techReopenTechFile

```
techReopenTechFile(
     d_techFileID
     t_mode
     )
     => t/nil
```

Description

Changes the mode of a technology file that has been opened. Use this function to upgrade the mode from `r` (read only) to a (append).

Caution

Mode changes other than from read to append are not recommended. If you change the mode from read or append to write, the contents of both the disk file and virtual memory are deleted. If you change the mode from append or write to read, the file contents are not refreshed and you are not prompted to save your changes when you exit.

Arguments

d_techFileID	The database identifier of the technology file.
t_mode	The mode in which to reopen the file. Valid Values: `a` (append or edit mode), `r` (read only), `w` (delete contents and load empty file); for recommended usage, refer to the Caution statement above

Return Values

`t`	The file was reopened in the specified mode.
`nil`	The technology file does not exist or the technology file is already open in append or write mode.

Example

```
techReopenTechFile( techFileID a )
=> t
```

Reopens the technology file identified by `techFileID` in `a` mode so you can edit it.

techCopyTechFile

```
techCopyTechFile(
     d_techFileID
     t_newTechLibName
     t_path
     [g_deleteOriginal]
     )
     => d_newTechFileID/nil
```

Description

Copies a technology file to a new location and optionally deletes the original. This function is especially useful for separating technology data out of libraries that contain both design data and technology data.

Arguments

d_techFileID	The database identifier of the technology file you want to copy.
t_newTechLibName	The name of the new technology library to create.
t_path	The UNIX path of the directory where you want the new technology library to be created. This path is automatically added to your `cds.lib` file.
g_deleteOriginal	Indicates that you want the design data in the original library attached to the new technology library and the original technology data removed from the design library. Valid Values: `t`, `nil` Default: `nil`

Return Values

d_newTechFileID	The database identifier of the new technology file.
`nil`	The technology file does not exist.

Example

```
techCopyTechFile( techFileID "newTechLib" "/usr2/lukan/4.4data" t )
=> t
```

Copies the technology file to a new library called `newTechLib` and reattaches the contents of the library identified by `techFileID` to the new technology library.

techDeleteTechFile

```
techDeleteTechFile(
     d_techFileID
     )
     => t/nil
```

Description

Deletes a technology file. This function is useful for deleting redundant technology data from design libraries that have been updated to share one technology library.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

`t`	The technology data was deleted.
`nil`	The technology file does not exist.

Example

```
techDeleteTechFile( techFileID )
=> t
```

Deletes the technology file identified by `techFileID`.

techSaveTechFile

```
techSaveTechFile(
     d_techFileID
     )
     => t/nil
```

Description

Saves the specified technology file from virtual memory to the disk file from which it was opened.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

`t`	The save completed successfully.
`nil`	The virtual memory version of the technology file was not saved; the specified technology file identifier is invalid or the system was not able to write to the directory containing the disk file.

Example

```
techSaveTechFile( techFileID )
=> t
```

Saves the technology file identified by `techFileID`.

techCloseTechFile

```
techCloseTechFile(
     d_techFileID
     )
     => t/nil
```

Description

Changes the status of the technology file to closed and decrements the close count. The technology file is not purged from virtual memory until the system needs to use the memory. Internally, the system maintains a count of the number of times you open and close a specific technology file. The count increments when you open and decrements when you close. When the close count is 0 and the system needs more virtual memory, it purges the technology file from virtual memory.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

`t`	The technology file was closed.
`nil`	The technology file does not exist.

Example

```
techCloseTechFile( techFileID )
=> t
```

Closes the technology file identified by `techFileID`.

techPurgeTechFile

```
techPurgeTechFile(
     d_techFileID
     )
     => t/nil
```

Description

Deletes the file from virtual memory if the close count for the file is 0. Internally, the system maintains a count of the number of times you open and close a specific technology file. The count increments when you open and decrements when you close. When the close count is 0 and the system needs more virtual memory, it automatically purges the technology file to free the memory. This function lets you manually purge the technology file.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

`t`	The memory was purged.
`nil`	The technology file does not exist or the close count is greater than 0.

Example

```
techPurgeTechFile( techFileID )
=> t
```

Frees the memory allocated to the technology file identified by `techFileID`.

techRefreshTechFile

```
techRefreshTechFile(
     d_techFileID
     )
     => t/nil
```

Description

Deletes the technology file loaded in virtual memory and reloads the binary file stored on disk. The edit mode (append, read, or write) of the file remains the same. If you refresh a technology file open in append mode, any changes you made and did not save are lost. If you refresh a technology file open in write mode, the empty disk file is loaded into virtual memory.

Caution

All changes not saved to disk are deleted when you use this function.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

t	The technology file was refreshed.
nil	The technology file does not exist.

Example

```
techRefreshTechFile( techFileID )
=> t
```

Refreshes the technology file identified by techFileID.

techTruncateTechFile

```
techTruncateTechFile(
     d_techFileID
     )
     => t/nil
```

Description

Deletes the contents of the technology file stored on disk.

Caution

This function deletes data from disk. Use this function with extreme care.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

t	The technology file was truncated.
nil	The technology file does not exist.

Example

```
techTruncateTechFile( techFileID )
=> t
```

Truncates the technology file identified by techFileID.

techGetDefaultTechName

```
techGetDefaultTechName(
     )
     => t_filename/nil
```

Description

Returns the default name of the binary technology file.

Arguments

None.

Return Values

t_filename	The default name of the binary technology file.
nil	The function failed.

Example

```
techGetDefaultTechName( )
=> "techfile.cds"
```

Returns the default binary technology file name, `techfile.cds`.

techGetTechFile

```
techGetTechFile(
     {g_libID | d_cellviewID}
     )
     => d_techFileID/nil
```

Description

Returns the database identifier of the technology file attached to the specified DFII library or cellview. If the cellview does not identify a specific technology file, this function returns the database identifier for the technology file bound to the library of the cellview. A library must identify a technology file.

Arguments

g_libID	The design data identifier for the library. To get the design data identifier, use the `ddGetObj` function. For more information, refer to the *Design Framework II SKILL Functions Reference*.
d_cellviewID	The database identifier for the cellview.

Return Values

d_techFileID	The database identifier of the technology file.
`nil`	The library or cellview does not exist.

Example

```
techGetTechFile( libID )
=> techFileID
```

Returns the database identifier of the technology file bound to the library identified by `libID`.

techGetTechFileDdId

```
techGetTechFileDdId(
     d_techFileId
     )
     => g_ddtechFileId/nil
```

Description

Returns the design data identifier for the technology file identified by the specified *techFileId*. This function is a wrapper for the ddGetObj function specifically designed to return the design data identifier of a technology file.

For more information about design data identifiers and ddGetObj, refer to the *Design Framework II SKILL Functions Reference*.

Arguments

d_techFileId	The database identifier of the technology file.

Return Values

g_ddtechFileId	The design data identifier of the technology file.
nil	The technology file does not exist.

Example

```
ddtechFileId =
techGetTechFileDdId (techFileId)
```

Returns the design data identifier of the technology file identified by techFileId.

If you want to get the technology file design data identifier from a library or cellview, you can nest functions as shown in the following example:

```
techDdId =
techGetTechFileDdId ( techGetTechFile( ddGetObj( "myLib" )))
```

techGetDeviceTechFile

```
techGetDeviceTechFile(
     d_cellviewID
     )
     => d_techFileID/nil
```

Description

Returns the database identifier for the technology file bound to the specified device.

Arguments

d_cellviewID	The database identifier for the device cellview.

Return Values

d_techFileID	The database identifier of the technology file.
nil	The device cellview does not exist.

Example

```
techGetDeviceTechFile( cellviewID )
=> techFileID
```

Returns the database identifier of the technology file bound to the cellview identified by cellviewID.

techGetOpenTechFiles

```
techGetOpenTechFiles(
     )
     => l_techfiles/nil
```

Description

Returns a list of the database identifiers of the technology files that are open.

Arguments

None.

Return Values

l_techfiles	A list of database identifiers for the open technology files.
nil	There are no technology files open.

Example

```
techGetOpenTechFiles()
=> (techfile1ID techfile2ID techfile3ID)
```

Returns the database identifiers of open technology files.

techVerifyTechFileId

```
techVerifyTechFileId(
     d_ID
     )
     => t/nil
```

Description

Determines whether the specified database identifier is for a technology file.

Arguments

d_ID	The database identifier you want to check.

Return Values

t	The database identifier is that of a technology file.
nil	The identifier is invalid or does not belong to a technology file.

Example

```
techVerifyTechFileId( techFileID )
=> t
```

Indicates that the database identifier is that of a technology file.

techSetEvaluate

```
techSetEvaluate(
     g_value
     )
     => t/nil
```

Description

Sets an internal flag that indicates whether the `tcDumpTechFile` and `techGet` functions evaluate expressions or read the expression as a string. The system automatically sets this internal flag while it dumps a technology file so that expressions and controls are preserved. This function lets you manually set the flag.

For more information about using control parameters, refer to "Controls Class Functions" on page 66.

Arguments

g_value	Indicates whether you want expressions evaluated when a technology file is dumped to ASCII. Valid Values: `t` (evaluate), `nil` (read as expressions)

Return Values

`t`	The internal flag is set to the value you specify.
`nil`	The input is neither `t` nor `nil`.

Example

```
techSetEvaluate(nil)
```

Sets the internal flag, so that expressions are not evaluated when processed.

techSetTimeStamp

```
techSetTimeStamp(
     d_techFileID
     )
     => t/nil
```

Description

Updates the internal time stamp of the technology file to the current time. The time stamp is an integer representing the number of seconds elapsed since 00:00:00 GMT, January 1, 1970.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

t	The time stamp was updated.
nil	The technology file does not exist.

Example

```
techSetTimeStamp( "myTechFile" )
=> t
```

Updates the time stamp to the current time.

techGetTimeStamp

```
techGetTimeStamp(
     d_techFileID
     )
     => x_timeStamp/nil
```

Description

Returns the time stamp of the technology file loaded into virtual memory. The time stamp is updated every time you modify the technology file in virtual memory. The time stamp is an integer representing the number of seconds elapsed since 00:00:00 GMT, January 1, 1970.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

x_timeStamp	The number of seconds elapsed since 00:00:00 GMT, January 1, 1970.
nil	The technology file does not exist.

Example

```
techGetTimeStamp( "myTechFile" )
=> 10281600
```

Returns the time elapsed since the last update. In this example, the time is 10281600/ 60*60*24 = 119 days

Attach Functions

These functions let you attach, or bind, a specific technology file to a DFII library or cellview. You can also delete the binding.

The technology file binding is made up of two properties, `techLib` and `techFile`. You must set both properties for every DFII library. You can optionally set these properties on individual cellviews. The properties are defined as follows:

`techLib`	The name of the DFII library containing the technology file.
`techFile`	The name of the binary technology file.

Cellviews inherit the binding of their library unless you have specifically set the properties for the cellview.

Note: A technology file must reside in a DFII library.

techBindTechFile

```
techBindTechFile(
     g_ID
     [t_techLibName
     [t_techfileName
     [updateDev]]]
     )
     => t/nil
```

Description

Attaches the specified DFII library, cell, or cellview to the specified technology file by creating `techLib` and `techFile` properties. To get the design data identifier for a DFII library, use the `ddGetObj` function.

For more information about design data identifiers and `ddGetObj`, refer to the *Design Framework II SKILL Functions Reference*.

Arguments

g_ID	The design data identifier for the library, cell, or cellview to attach.
t_techLibName	The design data identifier for the DFII library. Default: `cdsDefTechLib`
t_techfileName	The name of the binary technology file. You must specify a technology library name for *t_techLibName* before you can use this argument. Default: `techfile.cds`
updateDev	Indicates whether you want the system to update the binding for all instances in the design library. You must specify *t_techLibName* and *t_techfileName* before you can use this argument. Valid Values: `t`, `nil` Default: `nil`

Return Values

`t`	The `techFile` and `techLib` properties are set to the specified technology file.

`nil`	The library, cell, or cellview; or the technology library; or the technology file does not exist.

Example

```
techBindTechFile( ddGetObj("myLib")
"newTechLib" "techfile.cds" t )
=> t
```

Sets the `techLib` and `techFile` properties for the DFII library identified by `myLib` to `newTechLib` and `techfile.cds`. This example also updates all device instances in the design library to point to the devices in the technology library `newTechLib`.

techSetTechLibName

```
techSetTechLibName(
     g_ID
     t_libName
     )
     => t/nil
```

Description

Updates the techLib property of the specified library or cellview. This property is one of two that attach a library, cell, or cellview to a technology file.

Caution

Use this function only to update an existing techLib property. Use techBindTechFile to create the techLib and techFile properties.

Arguments

g_ID	The design data identifier for the library, cell, or cellview. To get the design data identifier, use ddGetObj. For more information, refer to the *Design Framework II SKILL Functions Reference*.
t_libName	The name of the DFII library containing the technology file you want to use.

Return Values

t	The techLib property is updated.
nil	The library, cellview, or technology file library name does not exist.

Example

```
techSetTechLibName( cellviewID ddGetObj("libName") )
=> t
```

Updates the techLib property for the cellview identified by cellviewID.

techSetTechFileName

```
techSetTechFileName(
     g_ID
     t_techfileName
     )
     => t/nil
```

Description

Updates the `techFile` property of the specified library or cellview. This property is one of two that attach a library or cellview to a technology file.

Caution

Use this function only to update an existing techFile property. Use techBindTechFile to create the techLib and techFile properties.

Arguments

g_ID	The design data identifier for the library, cell, or cellview. To get the design data identifier, use `ddGetObj`. For more information, refer to the *Design Framework II SKILL Functions Reference*.
t_techfileName	The name of the binary technology file. Valid Value: `techfile.cds`

Return Values

`t`	The `techFile` property is updated.
`nil`	The library, cellview, or technology file library name does not exist.

Example

```
techSetTechFileName(ddGetObj("libName") techfile.cds)
=> t
```

Updates the `techFile` binding property for the `libName` library to `techfile.cds`.

techGetTechLibName

```
techGetTechLibName(
     g_ID
     )
     => t_name/nil
```

Description

Returns the value of the `techLib` property set for the specified library or cellview. This property is one of two that attach a library or cellview to a technology file.

Arguments

g_ID	The design data identifier for the library, cell, or cellview. To get the design data identifier, use `ddGetObj`. For more information about design data identifiers and `ddGetObj`, refer to the *Design Framework II SKILL Functions Reference*.

Return Values

t_name	The value of the `techLib` property.
`nil`	The library, cellview, or technology library name does not exist.

Example

```
techGetTechLibName( ddGetObj("libName") )
=> techLib
```

Returns the name of the library bound to the library `libName`.

techGetTechFileName

```
techGetTechFileName(
     g_ID
     )
     => t_name/nil
```

Description

Returns the value of the `techFile` property for the specified DFII library or cellview. This property is one of two that attach a library or cellview to a technology file.

Arguments

g_ID	The design data identifier for the library, cell, or cellview. To get the design data identifier, use `ddGetObj`. For more information, refer to the *Design Framework II SKILL Functions Reference*.

Return Values

t_name	The value of the `techFile` property.
`nil`	The library or cellview does not exist.

Example

```
techGetTechFileName(ddGetObj("libName"))
=> techfile.cds
```

Returns the name of the technology file bound to the DFII library identified by the design data identifier obtained by `ddGetObj("libName")`.

techDeleteTechLibName

```
techDeleteTechLibName(
     g_ID
     )
     => t_name/nil
```

Description

Deletes the `techLib` property for the specified DFII library or cellview. This property is one of two that attach a library or cellview to a technology file.

Caution

DFII applications cannot display a cellview without a technology file. If you delete part of the technology file binding, DFII uses the default technology file <install_dir>/tools/dfII/samples/techfile/default.cds.

Arguments

g_ID	The design data identifier for the library, cell, or cellview. To get the design data identifier, use `ddGetObj`. For more information about design data identifiers and `ddGetObj`, refer to the *Design Framework II SKILL Functions Reference*.

Return Values

`t`	The `techLib` property is deleted.
`nil`	The library or cellview does not exist.

Example

```
techDeleteTechLibName(ddGetObj("libName"))
=> t
```

Deletes the `techLib` property for the library identified by the design data identifier obtained by `ddGetObj("libName")`.

techDeleteTechFileName

```
techDeleteTechFileName(
     g_ID
     )
     => t/nil
```

Description

Deletes the `techFile` property for the specified DFII library or cellview. This property is one of two that attach a library or cellview to a technology file.

Caution

If you delete the techFile property from a library, DFII uses the default technology file <install_dir>/tools/dfII/samples/techfile/default.cds.

Arguments

g_ID	The design data identifier for the library, cell, or cellview. To get the design data identifier, use `ddGetObj`. For more information, refer to the *Design Framework II SKILL Functions Reference*.

Return Values

`t`	The `techFile` property is deleted.
`nil`	The library or cellview does not exist.

Example

```
techDeleteTechFileName(ddGetObj("libName"))
=> t
```

Deletes the `techFile` property for the library identified by the design data identifier obtained by `ddGetObj("libName")`.

techUnattachTechFile

```
techUnattachTechFile(
     d_object
     )
     => t/nil
```

Description

Unattaches a technology library from a design library, cell, or cellview.

Caution

If you unattach the technology library from a design library, cell, or cellview without attaching another technology library to the design element, DFII uses the default technology file <install_dir>/tools/dfII/samples/techfile/default.cds.

Arguments

d_object	The design data identifier for the library, cell, or cellview for which to unattach its attached technology library. To get the design data identifier, use `ddGetObj`. For more information, refer to the *Design Framework II SKILL Functions Reference*.

Return Values

`t`	The technology library is unattached from the design object.
`nil`	The library, cell, or cellview does not exist.

Example

```
techUnattachTechFile(ddGetObj("libName"))
=> t
```

Unattaches the technology library currently attached to the library identified by `libID`.

Dump, Load, and Trigger Functions

The functions in this section let you transfer technology data from ASCII format to virtual memory (load) and back again (dump). You can also set programs to execute based on a dump or load trigger.

The dump and load trigger functions notify you when a dump or load event occurs. Triggers are of two types, preevent and postevent. Preevent triggers notify you before the event occurs, postevent triggers notify you afterward.

For example, you can use the predump trigger to append to a log file or add a comment header to the ASCII technology file. Or you can use the postload trigger to refresh custom menus or commands that need to be updated when technology data changes.

tcDumpTechFile

```
tcDumpTechFile(
     d_techFileID
     t_dumpFile
     )
     => t/nil
```

Description

Dumps the technology data in the specified technology file to an ASCII file.

Caution

If the specified dump file exists, this function overwrites it without warning. Cadence recommends that you use a temporary dump file and not overwrite your existing ASCII technology file.

Arguments

d_techFileID	The database identifier of the technology file to dump.
t_dumpFile	Name of the file to create.

Return Values

t	The dump completed successfully.
nil	The technology file is not open in virtual memory, or you do not have write permission for the dump file.

Example

```
tcDumpTechFile(techFileID "dumpFile")
```

Dumps the contents of the technology file identified by `techFileID` to a text file called `dumpFile`.

tcLoadTechFile

```
tcLoadTechFile(
     d_techFileID
     t_sourceFile
     [t_mode [l_classes]]
     )
     => t/nil
```

Description

Compiles the ASCII source file, opens the technology file in a (append) mode, and updates it with the compiled data. The default action is to merge the newly compiled data with the technology file already loaded into virtual memory. If you set *t_mode* to `w`, the system deletes the data in virtual memory and loads the newly compiled data. If you specify the classes to compile, this function compiles only the specified classes and preserves the data in the other classes.

If you opened the technology file in read mode, this function upgrades the mode to append or write. If you opened the technology file in append or write mode and then ran `tcLoadTechFile` with the default merge action, this function merges the newly compiled data with the data, if any, currently in virtual memory.

Arguments

d_techFileID	Database identifier of the technology file to compile.
t_sourceFile	Name of the ASCII technology file you want to compile and load. Valid Value: any string
t_mode	Indicates whether you want to replace the existing technology data in virtual memory with the newly compiled data. Valid Values: `w` replaces the data, a merges the data with the data that exists in virtual memory Default: a
l_classes	List of classes to compile. Valid Values: `controls`, `layerDefinitions`, `layerRules`, `devices`, `physicalRules`, `electricalRules`, `leRules`, `lxRules`, `compactorRules`, `prRules`

Return Values

`t`	The ASCII source file compiled successfully.
`nil`	The technology file does not exist or the source file does not exist, does not compile, or does not merge with the existing technology file properly. Error messages may be displayed listing syntax problems in the source file.

Example

```
tcLoadTechFile(techGetTechFile(cellviewID) "my_source")
```

Compiles the `my_source` ASCII file. Merges the technology data into the virtual memory used by the technology file for the cellview identified by `cellviewID`.

tcRegPreDumpTrigger

```
tcRegPreDumpTrigger(
     s_function
     [x_priority]
     )
     => t/nil
```

Description

Registers a trigger function the system calls before dumping a technology file. The *s_function* argument is a SKILL procedure with three arguments: the database identifier of the technology file, the port (or print destination) for the dump file, and the name of the dump file.

Trigger functions are called in order of priority, with 0 having the highest priority. If you do not specify a priority argument, the default is the lowest priority (that is, it executes after any other trigger function that specifies a priority).

You can use this function to add header information to the dumped technology file. You can also use this function to dump the technology data into your preferred format and prevent the system-provided dumping.

Arguments

s_function

Symbol indicating the name of the trigger function. The format of the function is

```
function(d_techFileID p_port t_techFileName)
=> t/nil
```

d_techFileID is the database identifier of the technology file.

p_port is the port for the dump file. (You can use the `infile` function to get port data for a file.)

t_techFileName is the name of the dump file.

x_priority

Priority of this trigger function.
Valid Values: any integer

Return Values

`t`	The trigger function registers.
`nil`	The SKILL procedure does not exist or is incomplete; the trigger function did not register.

Example

```
procedure( MYPreDumpTrigger( techFileID techport
     techfile )
if( techFileID~>owner == "sysAdmin"
  && techFileID~>MYTechName
  && techFileID~>MYversion then
     ; write header info at top of dumped file.
     fprintf(techport "Tech Name = %s.\nVer. %f\n"
     techFileID~>MYTechName techFileID~>MYversion)
    )
    t
)

tcRegPreDumpTrigger('MYPreDumpTrigger)
```

Registers a predump trigger that writes the header information at the top of the dumped technology file.

tcUnregPreDumpTrigger

```
tcUnregPreDumpTrigger(
     s_function
     )
     => t/nil
```

Description

Unregisters the specified predump trigger function so the system does not call it before dumping a technology file.

Arguments

s_function	Name of the registered trigger function (a symbol) to unregister.

Return Values

`t`	The trigger function registration is deleted.
`nil`	The trigger function registration does not exist.

Example

```
tcUnregPreDumpTrigger('MYPreDumpTriggerFunc)
```

The `MYPreDumpTriggerFunc` trigger function no longer executes.

tcRegPostDumpTrigger

```
tcRegPostDumpTrigger(
     s_function
     [x_priority]
     )
     => t/nil
```

Description

Registers a trigger function the system calls after it dumps a technology file. The *s_function* argument is a SKILL procedure with three arguments: the database identifier of the technology file, the port (or print destination) for the dump file, and the name of the dump file.

Trigger functions are called in order of priority, with 0 having the highest priority. If you do not specify a priority argument, the default is the lowest priority (that is, it is executed after any other trigger function that specifies a priority).

Arguments

s_function — Symbol indicating the name of the trigger function. The format is

```
function(d_techFileID p_port t_techFileName)
=> t/nil
```

d_techFileID is the database identifier of the technology file.

p_port is the port for the dump file. (You can use the `infile` function to get port data for a file.)

t_techFileName is the name of the dump file.

x_priority — Priority of this trigger function.
Valid Values: any integer

Return Values

`t` — The trigger function is registered.

`nil` — The SKILL procedure does not exist or is incomplete; the trigger function did not register.

tcUnregPostDumpTrigger

```
tcUnregPostDumpTrigger(
     s_function
     )
     => t/nil
```

Description

Unregisters the specified postdump trigger function so the system does not call it after dumping a technology file.

Arguments

s_function	Symbol indicating the name of the registered trigger function.

Return Values

`t`	The trigger function registration is deleted.
`nil`	The trigger function registration does not exist.

Example

```
tcUnregPostDumpEchFileTrigger('MYPostLoadTriggerFunc)
```

The `MYPostLoadTriggerFunc` trigger function no longer executes.

tcRegPreLoadTrigger

```
tcRegPreLoadTrigger(
     s_function
     [x_priority]
     )
     => t/nil
```

Description

Registers a trigger function the system calls before loading a technology file. The *s_function* argument is a SKILL procedure with two arguments: the database identifier of the technology file and the name of the ASCII technology you are loading.

Trigger functions are called in order of priority, with 0 having the highest priority. If you do not specify a priority argument, the default is the lowest priority (that is, it executes after any other trigger function that specifies a priority).

You can use this function to check the technology file contents before loading the technology file. Library administrators can use this mechanism to prevent library users from accidentally overwriting technology data that must be consistent with the centrally defined technology data.

Arguments

s_function	Symbol indicating the name of the trigger function. The format of the function is *function*(*d_techFileID t_techFileName*) => t/nil *d_techFileID* is the database identifier of the technology file. *t_techFileName* is the name of the ASCII technology file.
x_priority	Priority of this trigger function. Valid Values: any integer

Return Values

t	The trigger function is registered.
nil	The SKILL procedure does not exist or is incomplete; the trigger function did not register.

Example

```
procedure( MYFilterTechFileLoad( TFID techfile )
     prog( ( techport )
          if( TFID~>owner != "sysAdmin" return(t))
          ; TFID is owned by central admintrator.
          ; Do not allow user to overwrite protected
          ; technology data.
          techport = infile(techfile)
          ; Read file into big list then check the
          ; section headers.
          while( (section = read(techport))
               if( listp(section) && memq(car(section)
                    MYProtectedSections) then
                    warn("Cannot load techfilecontaining
                         \"%s\" section" car(section))
          close (techport)
                    return(nil)
               )
          )
          close(techport)
               return(t)
     )
)
tcRegPreLoadTrigger('MYFilterTechFileLoad)
```

Registers a preload trigger that prevents a user from accidentally overwriting the technology data defined by the library owner.

tcUnregPreLoadTrigger

```
tcUnregPreLoadTrigger(
     s_function
     )
     => t/nil
```

Description

Unregisters the specified preload trigger function so the system does not call it before loading a technology file.

Arguments

s_function	Name of the registered trigger function (a symbol) to unregister.

Return Values

`t`	The trigger function registration is deleted.
`nil`	The trigger function registration does not exist.

Example

```
tcUnregPreLoadTrigger('MYPreLoadTriggerFunc)
```

The `MYPreLoadTriggerFunc` trigger function no longer executes.

tcRegPostLoadTrigger

```
tcRegPostLoadTrigger(
     s_function
     [x_priority]
     )
     => t/nil
```

Description

Registers a trigger function the system calls after loading a technology file. The *s_function* argument is a SKILL procedure with two arguments: the database identifier of the technology file and the name of the ASCII technology file you are loading.

Trigger functions are called in order of priority, with 0 having the highest priority. If you do not specify a priority argument, the default is the lowest priority (that is, it executes after any other trigger function that specifies a priority).

You can use this function to keep track of the contents of the technology file or to update any related technology data based on the contents of the technology file.

Arguments

s_function	Symbol indicating the name of the trigger function. The format is *function*(*d_techFileId t_techFileName*) => t/nil *d_techFileId* is the database identifier of the technology file. *t_techFileName* is the name of the ASCII technology file.
x_priority	Priority of this trigger function. Valid Values: any integer

Return Values

t	The trigger function is registered.
nil	The SKILL procedure does not exist or is incomplete; the trigger function did not register.

Example

```
procedure( MYPostLoadTriggerFunc( techFileId techfile )
     if( techFileId~>owner == "sysAdmin" && techFileId~>mode == "a"
     then
     ; set flag to indicate that tech data has
     ; been overwritten
     techFileId~>myTechDataHasBeenModifiedFlag = t
     )
)
tcRegPostLoadTrigger('MYPostLoadTriggerFunc)
```

Registers a trigger function that indicates that the technology data in the technology library has been overwritten.

tcUnregPostLoadTrigger

```
tcUnregPostLoadTrigger(
     s_function
     )
     => t/nil
```

Description

Unregisters the specified trigger function so the system does not call it after loading a technology file.

Arguments

s_function	Symbol indicating the name of the registered trigger function to unregister.

Return Values

`t`	The trigger function registration is deleted.
`nil`	The trigger function registration does not exist.

Example

```
tcUnregPostLoadTrigger('MYPostLoadTriggerFunc)
```

The `MYPostLoadTriggerFunc` trigger function no longer executes.

Floating-Point Precision Function

The function in this section lets you set the precision, or number of digits following the decimal point, in floating-point numbers. Different systems can require different levels of precision for floating-point calculations.

techSetPrecision

```
techSetPrecision(
     x_digits
     )
     => t/nil
```

Description

Sets the precision, or number of digits following the decimal point, to be used in floating-point calculations.

Arguments

x_digits	Number of digits following the decimal point in floating-point numbers.

Return Values

`t`	The requested precision was set.
`nil`	The precision was not successfully set because the requested precision value was invalid.

Example

```
techSetPrecision (6)
```

This example sets the precision for floating-point numbers to 6.

Controls Class Functions

The Controls class lets you set global parameters and permissions that affect all classes of the technology file.

The `techParams` subclass of the Controls class lets you set parameters that you can use in the other classes. The `techPermissions` subclass of the Controls class lets you set read-only and read/write permissions on classes in your technology file for different system users. Users can then access technology file classes only as their assigned permissions allow.

The following example illustrates the use of `techParams`; it defines a parameter that is then used in two rules classes:

```
;************************************
; CONTROLS
;************************************
controls(
     techParams(lambda 0.8)
)
;*******************************
; PHYSICAL RULES
;*******************************
physicalRules(
     spacingRules(
     (minSpacing metal1 techGetParam(“lambda”))
     )
)
;*******************************
; COMPACTOR RULES
;*******************************
symRules(
(drc(("pimplant" "drawing" "PTAP") ("diff" "drawing" "NTR") (sep <
techGetParam("lambda"))))
)
```

When you use the `techParams` function in a rules class, the technology file compiler makes an explicit reference to the Controls class instead of evaluating the expression and storing only the value. When you dump the rules to a dump file, the control parameters, rather than the evaluated expressions, appear in the ASCII syntax.

If, at a later date, you want to update the rules using control parameters, you only need to update the value of the parameter in the Controls class.

Using Control Parameters with techSetEvaluate

In the following example, `techGetSpacingRule` returns the value of the control parameter when `techSetEvaluate` is set with the default `t` value. After you set the evaluation flag to `nil`, `techGetSpacingRule` returns the unevaluated value for the `minWidth` spacing rule.

Controls class portion of the loaded technology file

```
controls(
         techParams(
                     ("WIDTH" 0.8)
                     ("delta" 2.0)
                    )
         ) ;controls
```

Entered in the CIW

```
techSetSpacingRule(
         techID "minWidth"
         techGetParam("WIDTH") + 3.0) "metal1")
techSetEvaluate(t)
techGetSpacingRule(techID "minWidth")
```

Output in the CIW

```
3.8
```

Entered in the CIW

```
techSetEvaluate(nil)
techGetSpacingRule(techID "minWidth")
```

Output in the CIW

```
(techParams("WIDTH") + 3.0)
```

techSetParam

```
techSetParam(
     d_techFileID
     t_paramName
     g_paramValue
     )
     => t/nil
```

Description

Updates the value of the specified control parameter. If the parameter does not exist, this function creates it.

Arguments

d_techFileID	The database identifier of the technology file.
t_paramName	The name of the control parameter to update or create. Valid Values: any string
g_paramValue	The value to set for the parameter. Valid Values: any legal data type

Return Values

`t`	The control parameter was updated or created.
`nil`	The technology file does not exist.

Example

```
techSetParam("myTechFile" "lambda" 0.65)
=> t
```

Updates the value of the `lambda` control parameter in the `myTechFile` technology file.

techGetParams

```
techGetParams(
     d_techFileID
     )
     => l_params/nil
```

Description

Returns the list of control parameters that you have set in the specified technology file.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_params	List of control parameters and their values. The list has the following syntax: `( (t_paramName g_paramValue) ...)` *t_paramName* is the name of the control parameter. *g_paramValue* is the value of the parameter.
nil	The technology file does not exist, or the technology file does not define any control parameters.

Example

```
techGetParams( "myTechFile")
=> (("lambda" 0.6) ("theta" 2.5))
```

Returns the parameters defined in the myTechFile technology file.

techGetParam

```
techGetParam(
     d_techFileID
     t_name
     )
     => g_paramValue/nil
```

Description

Returns the value of the specified control parameter.

Arguments

d_techFileID	The database identifier of the technology file.
t_name	Name of the parameter for which you want the value.

Return Values

g_paramValue	The value of the specified control parameter.
nil	The technology file does not exist, or the specified parameter is not defined.

Example

```
techGetParam( "myTechFile" "lambda")
=> 0.6
```

Returns the value defined for the `lambda` control parameter in the `myTechFile` technology file.

techGetPermissions

```
techGetPermissions(
     d_techFileId
     )
     => l_permissions/nil
```

Description

Returns the permissions applied to the specified technology file.

Arguments

d_techFileId	The database identifier of the technology file.

Return Values

l_permissions	A list of the permissions set on the technology file.
nil	The technology file does not exist, or no permissions are set on it.

Example

The following SKILL functions get the database identifier for the the technology file attached to the current cellview:

```
cvID = deGetCellView()
db: 29839404
tfID = techGetTechFile(cvID)
db: 20926092
```

The following SKILL function lists the permissions on that technology file:

```
techGetPermissions(tfID)
      (("layerDefinitions"
              ("joe")
              nil
          )
          ("devices"
              ("joe")
              ("mary")
          )
      )
```

The software returns the permissions for any class with read-only or read/write permissions explicitly set. The following defines the format in which the software returns the information:

```
( ( className1
     (list_of_read-only_names) | nil
     (list_of_read/write_names) | nil
    )
    (className2
     (list_of_read-only_names) | nil
     (list_of_read/write_names) | nil
    )
    )
```

Note: The software may return both the names of users with read-only permissions and the names of users with read/write permissions on the same line, particularly if one of them is `nil`, but it always returns the names of users with read-only permissions first, followed by the names of users with read/write permissions.

In this example, the `techGetPermissions` SKILL command gets and displays the permissions assigned to the technology file with the database identifier stored in `tfID`. Two classes in this technology file have permissions explicitly assigned. For the `layerDefinitions` class,

- User `joe` has read-only permission
- As indicated by `nil`, no users are specifically assigned read/write permission, although all users except `joe` have read/write permission by default

For the `devices` class,

- User `joe` has read-only permission
- User `mary` is specifically assigned read/write permission
- All other users have read/write permission by default

Because no other classes have permissions explicitly assigned, all users have read/write access to them.

techGetPermission

```
techGetPermission(
     d_techFileId
     t_classname
     )
     => l_permissions/nil
```

Description

Returns the permissions applied to the specified class in the specified technology file.

Arguments

d_techFileId	The database identifier of the technology file.
t_classname	The name of the technology file class.

Return Values

l_permissions	A list of the permissions set on the specified class in the technology file.
nil	The technology file does not exist, the class does not exist, or no permissions are set on it.

Example

The following SKILL functions get the database identifier for the the technology file attached to the current cellview:

```
cvID = deGetCellView()
db: 29839404
tfID = techGetTechFile(cvID)
db: 20926092
```

The following SKILL function lists the permissions on the `layerDefinitions` class in that technology file:

```
techGetPermission(tfID "layerDefinitions")
     => (("joe") nil)
```

The first SKILL command returns the database identifier for the current cellview and stores it in the user-specified variable `cvID`. The second SKILL command returns the database

identifier for the technology file attached to the current cellview and stores it in the user-specified variable `tfID`. The final SKILL command gets and displays the permissions assigned to the `layerDefinitions` class in the technology file with the database identifier stored in `tfID`; it displays the names of users with read-only permission first (`joe`), followed by the names of users with read/write permission (`nil`).

techSetReadPermission

```
techSetReadPermission(
     d_techFileId
     t_classname
     t_rUser
     )
     => t/nil
```

Description

Sets read permission on the specified class in the specified technology file for the specified user.

Arguments

d_techFileId	The database identifier of the technology file.
t_classname	The name of the technology file class.
t_rUser	The name of the user.

Return Values

t	The read permission was set.
nil	The technology file does not exist.

Example

The following SKILL functions get the database identifier for the the technology file attached to the current cellview:

```
cvID = deGetCellView()
db: 29839404
tfID = techGetTechFile(cvID)
db: 20926092
```

The following SKILL function sets read permission on the `layerDefinitions` class in that technology file:

```
techSetReadPermission(tfID "layerDefinitions" "joe")
     => t
```

The first SKILL command returns the database identifier for the current cellview and stores it in the user-specified variable `cvID`. The second SKILL command returns the database identifier for the technology file attached to the current cellview and stores it in the user-specified variable `tfID`. The final SKILL command sets read-only permission for user `joe` on the `layerDefinitions` class in the technology file with the database identifier stored in `tfID`; it displays `t` to indicate successful application of the permission.

techSetReadWritePermission

```
techSetReadWritePermission(
     d_techFileId
     t_classname
     t_rwUser
     )
     => t/nil
```

Description

Sets read/write permission on the specified class in the specified technology file for the specified user.

Arguments

d_techFileId	The database identifier of the technology file.
t_classname	The name of the technology file class.
t_rwUser	The name of the user.

Return Values

`t`	The read/write permission was set.
`nil`	The technology file does not exist.

Example

The following SKILL functions get the database identifier for the the technology file attached to the current cellview:

```
cvID = deGetCellView()
db: 29839404
tfID = techGetTechFile(cvID)
db: 20926092
```

The following SKILL function sets read/write permission on the `layerDefinitions` class in that technology file:

```
techSetReadWritePermission(tfID "layerDefinitions" "joe")
     => t
```

The first SKILL command returns the data base identifier for the current cellview and stores it in the user-specified variable `cvID`. The second SKILL command returns the data base identifier for the technology file attached to the current cellview and stores it in the user-specified variable `tfID`. The final SKILL command sets read/write permission for user `joe` on the `layerDefinitions` class in the technology file with the data base identifier stored in `tfID`; it displays `t` to indicate successful application of the permission.

techIsReadPermission

```
techIsReadPermission(
     d_techFileId
     t_classname
     )
     => t/nil
```

Description

Indicates whether the current user has read permission on the specified class in the specified technology file.

Arguments

d_techFileId	The database identifier of the technology file.
t_classname	The name of the technology file class.

Return Values

t	The user has read permission.
nil	The technology file does not exist.

Example

The following SKILL functions get the database identifier for the the technology file attached to the current cellview:

```
cvID = deGetCellView()
db: 29839404
tfID = techGetTechFile(cvID)
db: 20926092
```

The following SKILL function checks for read permission on the `layerDefinitions` class in that technology file,

```
techIsReadPermission(tfID "layerDefinitions")
     => t
```

The first SKILL command returns the database identifier for the current cellview and stores it in the user-specified variable `cvID`. The second SKILL command returns the database identifier for the technology file attached to the current cellview and stores it in the user-

specified variable `tfID`. The final SKILL command checks for read permission for the current user on the `layerDefinitions` class in the technology file with the database identifier stored in `tfID`; it displays `t` to indicate that the user has read permission.

2

Layers SKILL Functions

The Layers SKILL functions create, set, and get data from the Layer Definitions and Layer Rules classes of the technology file. In general, the functions in this section do the following:

- The create functions add sections to the technology file that do not exist or replace whole or partial sections that do exist.
- The set functions update existing data or add new data to existing sections of the technology file.
- The get functions return data from sections of the technology file or `nil` if the data does not exist.
- The delete functions remove a specified piece of data from a section of the technology file.

Defining Layer-Purpose Pairs

The Layer Definitions class defines the layer-purpose pairs you use to create designs using Cadence® Design Framework II (DFII) applications. A layer-purpose pair is defined by a set of attributes that determine how it appears on a display device. A layer-purpose–pair definition requires the following:

- The layer must be defined in the `techLayers` subclass of the technology file.

  You can create a new layer or use a system-reserved layer. For more information about creating layers, refer to “techCreateLayer” on page 83 or to "techLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

- The purpose must be defined in the `techPurposes` subclass of the technology file.

  You can create a new purpose or use a system-reserved purpose. For more information about creating purposes, refer to “techCreatePurpose” on page 93 or to "techPurposes" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

- The layer-purpose pair must be listed in priority order with the other layer-purpose pairs in the `techLayerPurposePriorities` subclass of the technology file.

  For more information about creating layer-purpose pairs, refer to “techCreateLP” on page 103, to “techSetLPAttr” on page 106, or to "techLayerPurposePriorities" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

- The display definition of the layer-purpose pair must be listed in the `techDisplays` subclass of the technology file.

  For more information about defining how the layer-purpose pair appears in a display device, refer to “techSetLPAttr” on page 106 or to "techDisplays" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Using Layer-Purpose Pairs as Arguments to SKILL

Many of the subclasses in the technology file and many associated Cadence SKILL language functions accept either a layer or layer-purpose pair as arguments. If you use only the layer name or number as an argument, the system assumes that all purposes of the layer not listed separately apply. For example:

```
equivalentLayers(
   ((metal1 drawing) "metalPower")
   (metal1 vdd gnd)
)
```

In this example, all layer-purpose pairs using the `metalPower` layer are equivalent to each other and to the `metal1` drawing layer-purpose pair. The `metalPower` layers are not equivalent to any other `metal1` layer-purpose pairs, however. In addition, all layer-purpose pairs of `metal1` are equivalent to each other and to all layer-purpose pairs of `vdd` and `gnd`.

techCreateLayer

```
techCreateLayer(
     d_techFileID
     x_layerNumber
     t_layerName
     [t_layerAbbrev]
     )
     => d_layerID/nil
```

Description

Creates a layer in the `techLayers` subclass in the specified technology file and returns the database identifier that is used internally to access the layer. The `techLayers` subclass is located in the Layer Definitions class; it lists the layers that can be matched with a purpose to create layer-purpose pairs. If a `techLayers` subclass does not exist, this function creates one with the specified data.

Applications that display layer names do not always have room to display the entire name. The optional abbreviation expands your control over what is displayed in narrow fields. Depending upon the width of the field for displaying the layer name, an application displays whichever of the following fits:

- The full layer name
- The layer name truncated to fit (if no abbreviation is specified)
- The abbreviation
- The abbreviation truncated to fit

For more information about the `techLayers` subclass of the technology file, refer to "techLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
x_layerNumber	The layer number. Valid Values: a unique integer between 0 and 255 (layers numbered from 128 to 255 are reserved for the system)
t_layerName	The layer name. Valid Values: any string

t_layerAbbrev	An optional abbreviation of the layer name. Valid Values: any string of seven characters or less

Return Values

d_layerID	The database identifier that is used internally to identify the technology file and layer.
`nil`	The technology file does not exist; the layer is not created.

Example

```
techCreateLayer(techFileID 15 "metal1" "met1")
```

Creates a layer with the layer number `15`, the layer name `metal1`, and the layer abbreviation `met1`.

techSetLayerName

```
techSetLayerName(
     d_techFileID
     tx_layer
     t_layerName
     )
     => t/nil
```

Description

Updates the name of the specified layer in the `techLayers` subclass of the specified technology file. The `techLayers` subclass is located in the Layer Definitions class; it lists the layers that can be matched with a purpose to create layer-purpose pairs.

Note: When specifying a layer name, consider the possibility of shortened names being displayed in selection windows. See “techCreateLayer” on page 83 for details.

For more information about the `techLayers` subclass of the technology file, refer to "techLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
tx_layer	The layer name or number.
t_layerName	The new layer name. Valid Values: any string

Return Values

`t`	The layer name was successfully updated.
`nil`	The technology file does not exist or the layer is not defined in the technology file.

Example

```
techSetLayerName(techFileID 16 "metal2")
```

Updates the name of layer number `16` to `metal2` in the technology file identified by `techFileID`. The data is located in the `techLayers` subclass of the Layer Definitions class.

techSetLayerAbbrev

```
techSetLayerAbbrev(
     d_techFileID
     tx_layer
     t_layerAbbrev
     )
     => t/nil
```

Description

Updates the abbreviation of the specified layer in the `techLayers` subclass of the specified technology file. The `techLayers` subclass is located in the Layer Definitions class; it lists the layers that can be matched with a purpose to create layer-purpose pairs. You can indicate the layer to update by specifying the layer name or number.

Note: When specifying a layer name abbreviation, consider the possibility of shortened names being displayed in selection windows. See “techCreateLayer” on page 83 for details.

For more information about the `techLayers` subclass of the technology file, refer to "techLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
tx_layer	The layer name or number. Valid Values: the name or number of an existing layer; a layer number must be an integer between 0 and 255
t_layerAbbrev	The new abbreviation of the layer name. Valid Values: any string of seven characters or less

Return Values

`t`	The layer abbreviation was successfully updated.
`nil`	The technology file does not exist or the layer is not defined in the technology file.

Example

```
techSetLayerAbbrev(techFileID 16 "met2")
```

Updates the abbreviation of the `metal2` layer to `met2` in the `techLayers` subclass of the Layer Definitions class of the technology file identified by `techFileID`.

techGetLayerName

```
techGetLayerName(
     d_techFileID
     x_layerNumber
     )
     => t_layerName/nil
```

Description

Returns the layer name of the specified layer number defined in the `techLayers` subclass of the specified technology file. The `techLayers` subclass is located in the Layer Definitions class; it lists the layers that can be matched with a purpose to create layer-purpose pairs.

For more information about the `techLayers` subclass of the technology file, refer to "techLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file defining the layer.
x_layerNumber	The layer number. Valid Values: an integer between `0` and `255`

Return Values

t_layerName	The layer name.
`nil`	The technology file does not exist or the layer is not defined in the technology file.

Example

```
techGetLayerName(techFileID 1)
=> "nwell"
```

Returns the layer name `nwell` for layer number `1`.

techGetLayerNum

```
techGetLayerNum(
     d_techFileID
     t_layerName
     )
     => x_layerNumber/nil
```

Description

Returns the layer number of the specified layer name defined in the `techLayers` subclass of the specified technology file. The `techLayers` subclass is located in the Layer Definitions class; it lists the layers that can be matched with a purpose to create layer-purpose pairs.

For more information about the `techLayers` subclass of the technology file, refer to "techLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file defining the layer.
t_layerName	The layer name.

Return Values

x_layerNumber	The layer number.
`nil`	The technology file does not exist or the layer is not defined in the technology file.

Example

```
techGetLayerNum(techFileID "nwell")
=> 1
```

Returns the layer number 1 for the layer `nwell`.

techGetLayerAbbrev

```
techGetLayerAbbrev(
     d_techFileID
     tx_layer
     )
     => t_layerAbbrev/nil
```

Description

Returns the abbreviation of the specified layer from the `techLayers` subclass of the specified technology file. The `techLayers` subclass is located in the Layer Definitions class; it lists the layers that can be matched with a purpose to create layer-purpose pairs.

For more information about the `techLayers` subclass of the technology file, refer to "techLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
tx_layer	The layer name or number. Valid Values: the name or number of an existing layer; a layer number must be an integer between 0 and 255

Return Values

t_layerAbbrev	The abbreviation of the layer name or, if there is no abbreviation for the layer, the layer name.
`nil`	The technology file does not exist or the layer is not defined in the technology file.

Example

```
techGetLayerAbbrev(techFileID 16 )
=> "met2"
```

Returns the abbreviation of the `metal2` layer defined in the `techLayers` subclass of the Layer Definitions class of the technology file identified by `techFileID`.

techDeleteLayer

```
techDeleteLayer(
     d_techFileID
     tx_layer
     )
     => t/nil
```

Description

Deletes the specified layer from the `techLayers` subclass in the specified technology file. The `techLayers` subclass is located in the Layer Definitions class; it lists the layers that can be matched with a purpose to create layer-purpose pairs.

Caution

This function does not purge all references to the layer from the technology file. If the layer is referenced in a rule definition in another subclass of the technology file, an error will occur when an application attempts to use that rule. To prevent these errors, you must delete all references to the layer name from the ASCII technology file and load it again.

For more information about the `techLayers` subclass of the technology file, refer to "techLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`tx_layer`	The name or number of the layer to delete.

Return Values

`t`	The layer was deleted.
`nil`	The technology file does not exist or the layer is not defined in the technology file.

Example

```
techDeleteLayer( techFileID "metal5" )
```

Deletes the `metal5` layer from the technology file identified by `techFileID`.

techCreatePurpose

```
techCreatePurpose(
     d_techFileID
     x_purposeNumber
     t_purposeName
     [t_purposeAbbrev]
     )
     => d_purposeID/nil
```

Description

Creates a purpose in the `techPurposes` subclass of the specified technology file and returns the database identifier that is used internally to access the purpose. The `techPurposes` subclass is located in the Layer Definitions class; it lists the purposes that can be matched with a layer to create layer-purpose pairs. If a `techPurposes` subclass does not exist, this function creates one with the specified data. This function.

Applications that display purpose names do not always have room to display the entire name. The optional abbreviation expands your control over what is displayed in narrow fields. Depending upon the width of the field for displaying the purpose name, an application displays whichever of the following fits:

- The full purpose name
- The abbreviation
- The first and last letters of the full purpose name

For more information about the `techPurposes` subclass of the technology file, refer to "techPurposes" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
x_purposeNumber	The purpose number. Valid Values: a unique integer between 1 and 127
t_purposeName	The purpose name. Valid Values: any string
t_purposeAbbrev	The optional purpose abbreviation. Valid Values: any string of seven characters or less

Return Values

d_purposeID	The database identifier used internally to identify the technology file and purpose.
nil	The technology file does not exist; the purpose is not created.

Example

```
techCreatePurpose(techFileID 10 "drawing" "dwg")
```

Creates a purpose with the purpose number 10, the purpose name drawing, and the purpose abbreviation dwg.

techSetPurposeName

```
techSetPurposeName(
     d_techFileID
     x_purpose
     t_purposeName
     )
     => t/nil
```

Description

Updates the name of the specified purpose number in the specified technology file. The `techPurposes` subclass is located in the Layer Definitions class; it lists the purposes that can be matched with a layer to create layer-purpose pairs.

Note: When specifying a purpose name, consider the possibility of shortened names being displayed in selection windows. See “techCreatePurpose” on page 93 for details.

For more information about the `techPurposes` subclass of the technology file, refer to "techPurposes" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file defining the purpose.
x_purpose	The number of the purpose to update.
t_purposeName	The name to assign to the purpose. Valid Values: any string

Return Values

`t`	The purpose name was successfully updated.
`nil`	The technology file does not exist or the purpose is not defined in the technology file.

Example

```
techSetPurposeName( techFileID 5 "drawing5" )
```

Updates the name of purpose number 5 to `drawing5` in the technology file identified by `techFileID`.

techSetPurposeAbbrev

```
techSetPurposeAbbrev(
     d_techFileID
     tx_purpose
     t_purposeAbbrev
     )
     => t/nil
```

Description

Updates the abbreviation of the specified purpose in the specified technology file. The `techPurposes` subclass is located in the Layer Definitions class; it lists the purposes that can be matched with a layer to create layer-purpose pairs.

Note: When specifying a purpose abbreviation, consider the possibility of shortened names being displayed in selection windows. See “techCreatePurpose” on page 93 for details.

For more information about the `techPurposes` subclass of the technology file, refer to "techPurposes" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file defining the purpose.
tx_purpose	The purpose name or number.
t_purposeAbbrev	The abbreviation for the purpose name. Valid Values: any string of seven characters or less

Return Values

`t`	The purpose abbreviation was successfully updated.
`nil`	The technology file does not exist or the purpose is not defined in the technology file.

Example

```
techSetPurposeAbbrev(techFileID "drawing" "dwg")
```

Sets the abbreviation for the `drawing` purpose to dwg in the technology file identified by `techFileID`.

techGetPurposeName

```
techGetPurposeName(
     d_techFileID
     x_purposeNumber
     )
     => t_purposeName/nil
```

Description

Returns the purpose name of the specified purpose number defined in the `techPurposes` subclass of the specified technology file. The `techPurposes` subclass is located in the Layer Definitions class; it lists the purposes that can be matched with a layer to create layer-purpose pairs.

For more information about the `techPurposes` subclass of the technology file, refer to "techPurposes" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file defining the purpose.
x_purposeNumber	The purpose number.

Return Values

t_purposeName	The name of the purpose.
`nil`	The technology file does not exist or the purpose is not defined in the technology file.

Example

```
techGetPurposeName( techFileID 252 )
=> "drawing"
```

Returns the purpose `drawing` for purpose number `252`.

techGetPurposeNum

```
techGetPurposeNum(
     d_techFileID
     t_purposeName
     )
     => x_purposeNumber/nil
```

Description

Returns the purpose number of the specified purpose name from the `techPurposes` subclass of the specified technology file. The `techPurposes` subclass is located in the Layer Definitions class; it lists the purposes that can be matched with a layer to create layer-purpose pairs.

For more information about the `techPurposes` subclass of the technology file, refer to "techPurposes" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file defining the purpose.
`t_purposeName`	The purpose name.

Return Values

`x_purposeNumber`	The purpose number.
`nil`	The technology file does not exist or the purpose is not defined in the technology file.

Example

```
techGetPurposeNum( techFileID "drawing" )
=> 252
```

Returns the purpose number `252` for the purpose `drawing`.

techGetPurposeAbbrev

```
techGetPurposeAbbrev(
     d_techFileID
     tx_purpose
     )
     => t_layerAbbrev/nil
```

Description

Returns the abbreviation of the specified purpose from the `techPurposes` subclass of the specified technology file. The `techPurposes` subclass is located in the Layer Definitions class; it lists the purposes that can be matched with a layer to create layer-purpose pairs.

For more information about the `techPurposes` subclass of the technology file, refer to "techPurposes" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
tx_purpose	The purpose name or number. Valid Values: the name or number of an existing purpose; a purpose number must be an integer between 0 and 127

Return Values

t_purposeAbbrev	The abbreviation of the purpose name or, if there is no abbreviation for the purpose, the purpose name.
`nil`	The technology file does not exist or the purpose is not defined in the technology file.

Example

```
techGetPurposeAbbrev(techFileID drawing )
=> "drwg"
```

Returns the abbreviation of the `drawing` purpose defined in the `techPurposes` subclass of the Layer Definitions class of the technology file identified by `techFileID`.

techDeletePurpose

```
techDeletePurpose(
     d_techFileID
     tx_purpose
     )
     => t/nil
```

Description

Deletes the specified purpose from the `techPurposes` subclass of the specified technology file. The `techPurposes` subclass is located in the Layer Definitions class; it lists the purposes that can be matched with a layer to create layer-purpose pairs.

Caution

This function does not purge all references of the purpose from the technology file. If a layer-purpose pair with this purpose is referenced in a rule definition in another subclass of the technology file, an error will occur when an application attempts to use that rule. To prevent these errors, you must delete all references to the purpose from the ASCII technology file and recompile it.

For more information about the `techPurposes` subclass of the technology file, refer to "techPurposes" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`tx_purpose`	The purpose name or number.

Return Values

`t`	The purpose was deleted.
`nil`	The technology file does not exist or the purpose is not defined in the technology file.

Example

```
techDeletePurpose(techFileID "drawing5" )
```

Deletes the purpose `drawing5` from the technology file identified by `techFileID`.

techCreateLP

```
techCreateLP(
     d_techFileID
     l_layerPurpose
     t_lpName
     )
     => d_LPID/nil
```

Description

Creates a named layer-purpose pair in the specified technology file. The layer-purpose pair is defined with defaults in the `techLayerPurposePriorities`, `techDisplays`, and `streamLayers` subclasses of the Layer Definitions class.

This function also creates a database object for the layer-purpose pair and returns the database identifier. You use the identifier with the `techSetLPAttr` and `techSetLPPacketName` functions to update the defaults and to specify how the layer-purpose pair appears in your designs. The layer and purpose you specify must be defined in the technology file before you can create a layer-purpose pair. If the technology file, layer, or purpose does not exist or the layer-purpose pair already exists, this function returns `nil`.

For more information about the `techLayerPurposePriorities` subclass of the technology file, refer to "techLayerPurposePriorities" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*. For more information about the `techDisplays` subclass of the technology file, refer to "techDisplays" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Defaults

The defaults set by this function are as follows:

Technology File Subclass	Argument	Default
techLayerPurposePriorities	Priority	`0`

Technology File Subclass	Argument	Default
techDisplays	Packet	`defaultPacket`
	Visibility	`t`
	Selectability	`t`
	Contributes to changed layer	`t`
	Drag enable	`t`
	Valid	`t`
streamLayers	Stream number	layer number
	Data type	`0`
	Translate	`t`

Arguments

d_techFileID The database identifier of the technology file.

l_layerPurpose A list of the layer name or number and purpose name or number. The list has the following syntax:

`list(` *tx_layer tx_purpose* `)`

t_lpName The name of the layer-purpose pair.

Return Values

d_LPID The database identifier for the layer-purpose pair.

`nil` The technology file does not exist or the layer or purpose is not defined in the technology file; the layer-purpose pair is not created.

Example

```
techCreateLP( techFileID ( "nwell" "drawing" ) "nwell1")
=> t db:18483792
```

Creates a layer-purpose pair named `newell1` with the `nwell` layer and the `drawing` purpose in the technology file identified by `techFileID`. Returns the database identifier, which in this case is `18483792`. Also updates technology file subclasses with information about the new layer-purpose pair as illustrated below:

```
techLayerPurposePriorities(
(nwell drawing)
      .
      .
      .
)
techDisplays
(nwell drawing defaultPacket t t t t t)
      .
      .
      .
)
streamLayers(
;(layer             streamNum           dataType       translate)
 (nwell             136                 0              t
      .
      .
      .
)
```

techSetLPAttr

```
techSetLPAttr(
     d_LPID
     l_layerAttributes
     )
     => t/nil
```

Description

Updates layer attributes for the specified layer-purpose pair in the technology file associated with the layer-purpose pair. The layer attributes are specified as a list and include the display priority, visibility, selectability, whether changes to the layer contribute to the Diva verification products changed layer, if the layer is visible when dragged, and validity. These attributes are contained in the `techDisplays` and `techLayerPurposePriorities` subclasses of the Layer Definitions class.

For information about the `techLayerPurposePriorities` subclass of the technology file, refer to "techLayerPurposePriorities" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*. For information about the `techDisplays` subclass of the technology file, refer to "techDisplays" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_LPID	The database identifier of the layer-purpose pair.
l_layerAttributes	A list of layer attribute values. The list has the following syntax: `list( x_priority [g_visible] [g_selectable] [g_contToChgLay] [g_dragEnable] [g_valid] )` *x_priority* is the display priority assigned to the layer-purpose pair. Layer-purpose pairs with higher priorities are displayed on top of layer-purpose pairs with lower priorities. Valid Values: an integer between 0 and one less than the total number of currently defined layer-purpose pairs **Note:** If a layer-purpose pair already exists with the specified priority, the system reorders the list by increasing the priority of layer-purpose pairs with equal or higher priority by 1. *g_visible* indicates whether the layer-purpose pair is visible in the display device defined for the display packet associated

with this layer-purpose pair.
Valid Values: `t`, `nil`

`g_selectable` indicates whether objects drawn with the layer-purpose pair are selectable.
Valid Values: `t`, `nil`

`g_contToChgLay` indicates whether the layer-purpose pair contributes to a changed layer while using Diva verification products.
Valid Values: `t`, `nil`

`g_dragEnable` indicates whether you can drag a shape created with the layer-purpose pair with the Virtuoso® Layout Editor commands (for example, *Move* and *Copy*).
Valid Values: `t`, `nil`

`g_valid` indicates whether the layer-purpose pair is listed in the layer selection window.
Valid Values: `t`, `nil`

Return Values

`t`	The attributes were successfully updated for the layer-purpose pair.
`nil`	The technology file or layer-purpose pair does not exist.

Example

```
techSetLPAttr( LPID list( 4 t t t t t ) )
```

Sets attributes for the layer-purpose pair `pdiff drawing`.

techSetLPPacketName

```
techSetLPPacketName(
     d_LPID
     t_packetName
     )
     => t/nil
```

Description

Updates the display packet assigned to the specified layer-purpose pair in the `techDisplays` subclass of the technology file associated with the layer-purpose pair. The display packet must be defined in the display resource file before you can assign it to a layer-purpose pair.

For more information about the `techDisplays` subclass of the technology file, refer to "techDisplays" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*. For more information about defining display packets in the display resource file, refer to the *Technology File and Display Resource File User Guide*.

Arguments

d_LPID	The database identifier of the layer-purpose pair.
t_packetName	The name of the display packet to assign to the layer-purpose pair.

Return Values

`t`	The display packet was successfully assigned to the layer-purpose pair.
`nil`	The technology file, display packet, or layer-purpose pair does not exist.

Example

```
techSetLPPacketName( LPID "redsolid_S")
```

Assigns the display packet `redsolid_S` to the layer-purpose pair identified by `LPID`.

techGetLP

```
techGetLP(
     d_techFileID
     l_layerPurpose
     )
     => l_LPID/nil
```

Description

Returns the database identifier of the specified layer-purpose pair.

Arguments

d_techFileID	The database identifier of the technology file.
l_layerPurpose	A list of the layer name or number and purpose name or number. The list has the following syntax: `list( tx_layer tx_purpose )`

Return Values

d_LPID	The database identifier of the layer-purpose pair.
`nil`	The technology file or layer-purpose pair does not exist.

Example

```
techGetLP( techFileID list("diff" "drawing") )
```

Returns the database identifier for the layer-purpose pair `diff drawing` defined in the technology file identified by `techFileID`.

techGetLPAttr

```
techGetLPAttr(
     d_LPID
     )
     => l_value/nil
```

Description

Returns the list of attributes for the specified layer-purpose pair. The attributes (display priority, visibility, selectability, whether changes to the layer contribute to the Diva verification products changed layer, if the layer is visible when dragged, and validity) are defined in the `techLayerPurposePriorities` and `techDisplays` subclasses of the technology file associated with the layer-purpose pair. If the layer-purpose pair does not exist, this function returns `nil`.

For more information about the `techLayerPurposePriorities` subclass of the technology file, refer to "techLayerPurposePriorities" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*. For more information about the `techDisplays` subclass of the technology file, refer to "techDisplays" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_LPID	The database identifier of the layer-purpose pair.

Return Values

l_value	A list of layer attributes. The list has the following syntax: `list( x_priority g_visible g_selectable g_contToChgLay g_dragEnable g_valid )` *x_priority* is the display priority assigned to the layer-purpose pair. *g_visible* indicates whether the layer-purpose pair is visible in the display device. *g_selectable* indicates whether objects drawn with the layer-purpose pair are selectable.

g_contToChgLay indicates whether the layer-purpose pair contributes to a changed layer while using Diva verification products.

g_dragEnable indicates whether you can drag a shape created with the layer-purpose pair in the Virtuoso Layout Editor.

g_valid indicates whether the layer-purpose pair appears in the LSW.

`nil` The technology file or layer-purpose pair does not exist.

Example

```
techGetLPAttr( LPID )
=> (4 t t t t t)
```

Returns the attributes assigned to the specified layer-purpose pair.

techGetLPPacketName

```
techGetLPPacketName(
     d_lpID
     )
     => t_packetName/nil
```

Description

Returns the name of the display packet defined for the specified layer-purpose pair. The display packet attribute is defined in the `techDisplays` subclass of the technology file associated with the layer-purpose pair.

For more information about the `techDisplays` subclass of the technology file, refer to "techDisplays" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_lpID	The database identifier of the layer-purpose pair.

Return Values

t_packetName	The name of the display packet assigned to the layer-purpose pair.
`nil`	The database identifier for the layer-purpose pair does not exist.

Example

```
techGetLPPacketName( LPID )
=> "redSolid_S"
```

Returns `redSolid_S`, the name of the display packet assigned to the layer-purpose pair identified by `LPID`.

techDeleteLP

```
techDeleteLP(
     d_LPID
     )
     => t/nil
```

Description

Deletes the specified layer-purpose pair from the technology file associated with the layer-purpose pair. The layer-purpose pair is defined in the Layer Definitions class.

Caution

This function only deletes the layer-purpose–pair definition from the Layer Definitions class. If the layer-purpose pair is referenced in other classes of the technology file, an error will occur when an application attempts to use the rule or device specifying the layer-purpose pair. To prevent these errors, you must delete all references of the layer-purpose pair from the ASCII technology file and load it again.

For more information about the Layer Definitions class, refer to "Layer Definitions Class" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

For more information about deleting layer-purpose pairs, refer to “Deleting a Layer” in the *Technology File and Display Resource File User Guide*.

Arguments

d_LPID	The database identifier of the layer-purpose pair to delete.

Return Values

`t`	The layer-purpose pair was deleted from the Layer Definitions class.
`nil`	The technology file does not exist or the layer-purpose pair is not defined in the technology file.

Example

```
techDeleteLP(LPID)
=> t
```

Deletes the specified layer-purpose pair from the Layer Definitions class. References to the layer-purpose pair might still exist in other technology file classes, which must be updated separately.

techSetEquivLayers

```
techSetEquivLayers(
     d_techFileID
     l_equivLayers
     )
     => t/nil
```

Description

Creates a set of equivalent layers in the `equivalentLayers` subclass of the specified technology file. The `equivalentLayers` subclass is located in the Layer Rules class; it lists layers that represent the same kind of material. If an `equivalentLayers` subclass does not exist, this function creates one with the specified data. If the technology file already defines equivalent layers, this function deletes and replaces the `equivalentLayers` subclass with the specified data.

For more information about the `equivalentLayers` subclass of the technology file, refer to "equivalentLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`l_equivLayers`	A list of lists indicating the equivalent layers to create. The list has the following syntax: `list( list( ltx_layer ...) ... )` *`ltx_layer`* is a layer you specify as equivalent to the other layers you list. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose

Return Values

`t`	The `equivalentLayers` subclass of the specified technology file was created or recreated.
`nil`	The technology file does not exist.

Example

```
techSetEquivLayers( techFileID list(
list("metal1" "vdd1" gnd1)
list("metal2" "vdd2" "gnd2")
list(list("metal1" "pin") "pinMetal")
)
```

Recreates the `equivalentLayers` subclass of the technology file identified by `techFileID` to define the specified equivalent layers.

techSetEquivLayer

```
techSetEquivLayer(
     d_techFileID
     l_equivLayers
     )
     => t/nil
```

Description

Appends the specified set of equivalent layers to the `equivalentLayers` subclass of the specified technology file. The `equivalentLayers` subclass is located in the Layer Rules class; it lists layers that represent the same kind of material.

For more information about the `equivalentLayers` subclass of the technology file, refer to "equivalentLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
l_equivLayers	A list of equivalent layers. The list has the following syntax: `list( ltx_layer ... )` *ltx_layer* is a layer you specify as equivalent to the other layers you list. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose

Return Values

`t`	The equivalent layers were appended to the `equivalentLayers` subclass of the technology file, or the specified layers were already listed as equivalent layers in the `equivalentLayers` subclass.
`nil`	The technology file does not exist or does not define any equivalent layers.

Example

```
techSetEquivLayer( techFileID list("metal1" "vdd") )
```

Appends the equivalent layer set of `metal1` and `vdd` to the `equivalentLayers` subclass of the technology file identified by `techFileID`.

techGetEquivLayers

```
techGetEquivLayers(
     d_techFileID
     )
     => l_equivLayersList/nil
```

Description

Returns the equivalent layers defined in the `equivalentLayers` subclass of the specified technology file. The `equivalentLayers` subclass is located in the Layer Rules class; it lists layers that represent the same kind of material.

For more information about the `equivalentLayers` subclass of the technology file, refer to "equivalentLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_equivLayersList	List of lists indicating which layers are equivalent. The list has the following syntax: `( (lt_layer ... ) ... )` *lt_layer* is an equivalent layer in the `equivalentLayers` subclass of the technology file. The layer is listed as it appears in the technology file. It can be a layer name or a layer-purpose pair.
`nil`	The technology file does not exist or does not define any equivalent layers.

Example

```
techGetEquivLayers( techFileID )
=> ( ("metal1" "vdd") ("metal2" "gnd") )
```

Returns the equivalent layers defined in the technology file identified by `techFileID`.

techSetViaLayers

```
techSetViaLayers(
     d_techFileID
     l_viaLayers
     )
     => t/nil
```

Description

Creates a set of via layers in the `viaLayers` subclass of the specified technology file. The `viaLayers` subclass is located in the Layer Rules class; it lists layers that conduct between two other layers. If a `viaLayers` subclass does not exist, this function creates one with the specified data. If the technology file already defines via layers, this function deletes and replaces the `viaLayers` subclass of the technology file with the specified data.

For more information about the `viaLayers` subclass of the technology file, refer to "viaLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
l_viaLayers	A list of lists indicating the via layers to create. The list has the following syntax: `list( list( ltx_bottom ltx_via ltx_top ) ... )` *ltx_bottom* is the bottom routing layer of the via. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose *ltx_via* is the middle layer of the via, commonly called the via layer. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose *ltx_top* is the top routing layer. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose

Return Values

`t`	The via layers were created or recreated.
`nil`	The technology file does not exist.

Example

```
techSetViaLayers( techFileID list(
list( "metal1" "via" "metal2")
list( "metal2" "via2" "poly")
))
```

Recreates the `viaLayers` subclass of the technology file identified by `techFileID` to define the specified via layers.

techSetViaLayer

```
techSetViaLayer(
     d_techFileID
     ltx_bottom
     ltx_via
     ltx_top
     )
     => t/nil
```

Description

Appends the specified set of via layers to the `viaLayers` subclass of the specified technology file. The `viaLayers` subclass is located in the Layer Rules class; it lists layers that conduct between two other layers.

For more information about the `viaLayers` subclass of the technology file, refer to "viaLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
ltx_bottom	The bottom routing layer of the via. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose
ltx_via	The middle layer of the via, commonly called the via layer. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose
ltx_top	The top routing layer. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose

Return Values

`t`	The via layers were appended to the `viaLayers` subclass of the technology file, or the specified layers were already defined in the `viaLayers` subclass.
`nil`	The technology file does not exist.

Example

```
techSetViaLayer( techFileID "metal1" "via" "metal2" )
```

Appends the specified via data to the `viaLayers` subclass of the technology file identified by `techFileID`.

techGetViaLayers

```
techGetViaLayers(
     d_techFileID
     )
     => l_viaLayers/nil
```

Description

Returns lists of the three layers defining each via in the `viaLayers` subclass of the specified technology file. The `viaLayers` subclass is located in the Layer Rules class; it lists layers that conduct between two other layers.

For more information about the `viaLayers` subclass of the technology file, refer to "viaLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_viaLayers	The list of via definitions. The list has the following syntax: `list( list( lt_bottom lt_via lt_top ) ... )` *lt_bottom* is the bottom routing layer of the via. Valid Values: the layer name, a list containing the layer name and layer purpose *lt_via* is the middle layer of the via, commonly called the via layer. Valid Values: the layer name, a list containing the layer name and layer purpose *lt_top* is the top routing layer. Valid Values: the layer name, a list containing the layer name and layer purpose
`nil`	The technology file does not exist or does not define any via layers.

Example

```
techGetViaLayers(techFileID)
=> (
("metal1" "via" "metal2")
("metal2" "via2" "poly")
)
```

Returns the layers making up the vias in the technology file identified by `techFileID`.

techGetOuterViaLayers

```
techGetOuterViaLayers(
     d_techFileID
     ltx_viaLayer
     )
     => l_outerViaLayers/nil
```

Description

Given the via layer, or the middle layer of the via, returns the bottom and top layers defined in the `viaLayers` subclass of the specified technology file. The `viaLayers` subclass is located in the Layer Rules class; it lists layers that conduct between two other layers.

For more information about the `viaLayers` subclass of the technology file, refer to "viaLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
ltx_viaLayer	The via layer, or middle layer of a via. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose

Return Values

l_outerViaLayers	A list of the bottom and top layers of the via identified by the via layer. The list has the following syntax: `list( lt_bottom lt_top )` *lt_bottom* is the bottom routing layer of the via. Valid Values: the layer name, a list containing the layer name and layer purpose *lt_top* is the top routing layer. Valid Values: the layer name, a list containing the layer name and layer purpose
`nil`	The technology file does not exist, or the specified layer is not a via layer.

Example

```
techGetOuterViaLayers( techFileID "via" )
=> ("metal1" "metal2")
```

Returns the layers surrounding the specified via layer in the technology file identified by `techFileID`.

techIsViaLayer

```
techIsViaLayer(
     d_techFileID
     ltx_viaLayer
     )
     => t/nil
```

Description

Indicates whether the specified layer is defined as a via layer (the middle layer of a via) in the `viaLayers` subclass of the specified technology file. The `viaLayers` subclass is located in the Layer Rules class; it lists layers that conduct between two other layers.

For more information about the `viaLayers` subclass of the technology file, refer to "viaLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`ltx_viaLayer`	The layer you want to check. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose

Return Values

`t`	The specified layer is a via layer.
`nil`	The technology file does not exist, or the specified layer is not a via layer.

Example

```
techIsViaLayer( techFileID "via" )
=> t
```

The `via` layer is the middle layer of a via defined in the `viaLayers` subclass of the technology file identified by `techFileID`.

techSetStreamLayer

```
techSetStreamLayer(
     d_techFileID
     ltx_layer
     g_streamNumber
     [g_datatype]
     [g_translate]
     )
     => t/nil
```

Description

Updates the Stream data for the specified layers in the `streamLayers` subclass of the specified technology file. The `streamLayers` subclass is located in the Layer Rules class; it contains the information required to translate your design to GDSII Stream format. If the `streamLayers` subclass does not exist, this function creates one with the specified data. If layer is not listed in the `streamLayers` subclass, this function appends the data.

For more information about the `streamLayers` subclass of the technology file, refer to "streamLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
ltx_layer	The layer for which to provide Stream translation data. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose

Note: If you specify a layer name or number, the stream number applies to all layer-purpose pairs using the specified layer unless you specify an individual layer-purpose pair separately.

g_streamNumber	The Stream number to assign to the layer. Valid Values: `0` through `255` Default: the layer number
g_datatype	The Stream datatype of the stream layer. Valid Values: `0` through `127` Default: `0`

g_translate	Indicates whether the layer should be translated. Valid Values: `t`, `nil` Default: `t`

Return Values

`t`	The Stream translation data was updated or created for the specified layer.
`nil`	The technology file does not exist.

Example

```
techSetStreamLayer( techFileID list("metal1" "drawing") 3 1 t)
```

Indicates that you want the `metal1` `drawing` layer-purpose pair translated to Stream layer number 3 and datatype `1`.

techSetStreamNumber

```
techSetStreamNumber(
     d_techFileID
     ltx_layer
     g_streamNumber
     )
     => t/nil
```

Description

Updates the Stream number assigned to the specified layer in the `streamLayers` subclass of the specified technology file.The `streamLayers` subclass is located in the Layer Rules class; it contains the information required to translate your design to GDSII Stream format.

For more information about the `streamLayers` subclass of the technology file, refer to "streamLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`ltx_layer`	The layer for which to provide Stream translation data. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose
`g_streamNumber`	The Stream number to assign to the layer. Valid Values: `0` through `255` Default: the layer number

Return Values

`t`	The Stream number was successfully updated.
`nil`	The technology file does not exist, or the specified layer is not listed in the `streamLayers` subclass of the technology file.

Example

```
techSetStreamNumber( techFileID "metal1" 3 )
```

Updates the Stream number of the `metal1` layer to 3.

techSetStreamDatatype

```
techSetStreamDatatype(
     d_techFileID
     ltx_layer
     g_datatype
     )
     => t/nil
```

Description

Updates the Stream datatype assigned to the specified layers in the `streamLayers` subclass of the specified technology file. The `streamLayers` subclass is located in the Layer Rules class; it contains the information required to translate your design to GDSII Stream format.

For more information about the `streamLayers` subclass of the technology file, refer to "streamLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
ltx_layer	The layer for which to provide Stream translation data. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose
g_datatype	The Stream datatype to assign to the layer. Valid Values: `0` through `127` Default: `0`

Return Values

`t`	The Stream datatype was successfully updated.
`nil`	The technology file does not exist, or the specified layer is not listed in the `streamLayers` subclass of the technology file.

Example

```
techSetStreamDatatype( techFileID "metal1" 0)
```

Updates the Stream datatype of the `metal1` layer to `0`.

techSetStreamTranslate

```
techSetStreamTranslate(
     d_techFileID
     ltx_layer
     g_translate
     )
     => t/nil
```

Description

Updates the Stream translation indicator assigned to the specified layer in the `streamLayers` subclass of the specified technology file. The `streamLayers` subclass is located in the Layer Rules class; it contains the information required to translate your design to GDSII Stream format.

For more information about the `streamLayers` subclass of the technology file, refer to "streamLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
ltx_layer	The layer for which to provide Stream translation data. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose
g_translate	Indicates whether the layer should be translated. Valid Values: `t`, `nil`

Return Values

`t`	The Stream translation indicator was successfully updated.
`nil`	The technology file does not exist, or the specified layer is not listed in the `streamLayers` subclass of the technology file.

Example

```
techSetStreamTranslate( techFileID "metal1" nil)
```

Updates the Stream translation indicator of the `metal1` layer to `nil`. The layer will not be translated when you use the `Stream Out` or `pipo` command to create a Stream file.

techGetStreamLayers

```
techGetStreamLayers(
     d_techFileID
     )
     => l_streamLayers/nil
```

Description

Returns the layers and associated Stream translation data defined in the `streamLayers` subclass of the specified technology file. The `streamLayers` subclass is located in the Layer Rules class; it contains the information required to translate your design to GDSII Stream format.

For more information about the `streamLayers` subclass of the technology file, refer to "streamLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_streamLayers	A list of lists containing the Stream translation data. The list has the following syntax: `list( list( lt_layer g_number g_datatype g_translate) ... )` *lt_layer* is either a list of the layer and purpose or the layer name. *g_number* is the Stream number assigned to the layer. *g_datatype* is the Stream datatype of the stream layer. *g_translate* indicates whether the layer should be translated.
`nil`	The technology file does not exist or does not contain any Stream translation data.

Example

```
techGetStreamLayers( techFileID )
=> (
( "metal1" 3 0 t )
( "metal2" 4 0 t )
)
```

Returns the Stream data specified in the technology file identified by `techFileID`.

techGetStreamLayer

```
techGetStreamLayer(
     d_techFileID
     ltx_layer
     )
     => l_streamData/nil
```

Description

Returns the Stream data for the specified layer in the `streamLayers` subclass of the specified technology file. The `streamLayers` subclass is located in the Layer Rules class; it contains the information required to translate your design to GDSII Stream format.

For more information about the `streamLayers` subclass of the technology file, refer to "streamLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
ltx_layer	The layer for which to return Stream translation data. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose

Return Values

l_streamData	A list containing the stream data for the specified layer. The list has the following syntax: `list( x_number x_datatype x_translate )` *x_number* is the Stream number assigned to the layer. *x_datatype* is the Stream datatype of the stream layer. *x_translate* indicates whether the layer should be translated.
`nil`	The technology file does not exist or does not contain Stream translation data for the specified layer.

Example

```
techGetStreamLayer( techFileID "metal1" )
=> list( 3 0 t )
```

Returns the Stream data specified for the `metal1` layer in the technology file identified by `techFileID`.

techGetStreamNumber

```
techGetStreamNumber(
     d_techFileID
     ltx_layer
     )
     => g_streamNumber/nil
```

Description

Returns the Stream number assigned to the specified layer in the `streamLayers` subclass of the specified technology file. The `streamLayers` subclass is located in the Layer Rules class; it contains the information required to translate your design to GDSII Stream format.

For more information about the `streamLayers` subclass of the technology file, refer to "streamLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
ltx_layer	The layer for which to return Stream translation data. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose

Return Values

g_streamNumber	The Stream number assigned to the layer.
`nil`	The technology file does not exist or does not contain Stream translation data for the specified layer.

Example

```
techGetStreamNumber( techFileID "metal1" )
=> 3
```

Returns the Stream number specified for the `metal1` layer in the technology file identified by `techFileID`.

techGetStreamDatatype

```
techGetStreamDatatype(
     d_techFileID
     ltx_layer
     )
     => g_datatype/nil
```

Description

Returns the Stream datatype assigned to the specified layer in the `streamLayers` subclass of the specified technology file. The `streamLayers` subclass is located in the Layer Rules class; it contains the information required to translate your design to GDSII Stream format.

For more information about the `streamLayers` subclass of the technology file, refer to "streamLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
ltx_layer	The layer for which to return Stream translation data. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose

Return Values

g_datatype	The Stream datatype assigned to the layer.
`nil`	The technology file does not exist or it does not contain Stream translation data for the specified layer.

Example

```
techGetStreamDatatype( techFileID "metal1" )
=> 0
```

Returns the Stream datatype specified for the `metal1` layer in the technology file identified by `techFileID`.

techGetStreamTranslate

```
techGetStreamTranslate(
     d_techFileID
     ltx_layer
     )
     => g_translate/nil
```

Description

Returns the Stream translation indicator assigned to the specified layer in the `streamLayers` subclass of the specified technology file. The `streamLayers` subclass is located in the Layer Rules class; it contains the information required to translate your design to GDSII Stream format.

For more information about the `streamLayers` subclass of the technology file, refer to "streamLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
ltx_layer	The layer for which to return Stream translation data. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose

Return Values

g_translate	The Stream translation indicator assigned to the layer.
`nil`	The technology file does not exist or does not contain Stream translation data for the specified layer.

Example

```
techGetStreamTranslate( techFileID "metal1" )
=> t
```

Returns the Stream translation indicator specified for the `metal1` layer in the technology file identified by `techFileID`. The `metal1` layer is translated when you use the `Stream Out` command to create a Stream file.

techSetLayerProp

```
techSetLayerProp(
     d_techFileID
     ltx_layer
     l_propertyValue
     )
     => t/nil
```

Description

Updates the value of the specified layer property in the `techLayerProperties` subclass of the specified technology file. The `techLayerProperties` subclass is located in the Layer Definitions class; it specifies special properties that you want to place on the layers in your design. If the `techLayerProperties` subclass does not exist, this function creates one with the specified data. If the property does not exist, the function creates a new layer property and sets the value to the specified value.

For more information about the `techLayerProperties` subclass of the technology file, refer to "techLayerProperties" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file defining the layer.
ltx_layer	A list containing the layer name and purpose, or the layer name, or the layer number of the layer.
l_propertyValue	A list specifying the property name and value. The list has the following syntax: `list( t_propName g_propValue )` *t_propName* is the name of the property Valid Values: any string *g_propValue* is the value of the property. Valid Values: an integer, a floating-point number, a string enclosed in quotation marks, a Boolean value, any SKILL symbol or expression that evaluates to any of these types

Return Values

`t`	The property has been updated or created successfully.
`nil`	The technology file or layer does not exist.

Example

```
techSetLayerProp(techFileID "metal1" list("myCorpCADControlValue" t))
=> t
```

Sets the `myCorpCADControlValue` property to `t` on the `metal1` layer in the technology file identified by `techFileID`.

techSetTwoLayerProp

```
techSetTwoLayerProp(
     d_techFileID
     ltx_layer1
     ltx_layer2
     l_propertyValue
     )
     => t/nil
```

Description

Updates the value of the specified two-layer property in the `techLayerProperties` subclass of the specified technology file. The `techLayerProperties` subclass is located in the Layer Definitions class; it specifies special properties that you want to place on the layers in your design. If the `techLayerProperties` subclass does not exist, this function creates one with the specified data. If the property does not exist, the function creates a new two-layer property and sets the value to the specified value.

For more information about the `techLayerProperties` subclass of the technology file, refer to "techLayerProperties" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file defining the layer.
ltx_layer1	The first layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
ltx_layer2	The second layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
l_propertyValue	A list specifying the property name and value. The list has the following syntax: `list( t_propName g_propValue )` *t_propName* is the name of the property. Valid Values: any string

g_propValue is the value of the property.
Valid Values: an integer, a floating-point number, a string enclosed in quotation marks, a Boolean value, any SKILL symbol or expression that evaluates to any of these types

Return Values

`t`	The property has been updated or created successfully.
`nil`	The technology file or layer does not exist.

Example

```
techSetTwoLayerProp(techFileID "metal1"  "metal2" list("myCorpCADControlValue" t))
=> t
```

Sets the `myCorpCADControlValue` property to `t` on the layers `metal1` and `metal2` in the technology file identified by `techFileID`.

techGetLayerProp

```
techGetLayerProp(
     d_techFileID
     ltx_layer
     t_propName
     )
     => g_propValue/nil
```

Description

Returns the value of the specified layer property from the `techLayerProperties` subclass of the specified technology file. The `techLayerProperties` subclass is located in the Layer Definitions class; it specifies special properties that you want to place on the layers in your design.

For more information about the `techLayerProperties` subclass of the technology file, refer to "techLayerProperties" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file defining the layer.
ltx_layer	A list containing the layer name and purpose, or the layer name, or the layer number of the layer.
t_propName	The name of the property.

Return Values

g_propValue	The value of the property.
`nil`	The technology file or layer does not exist or the property value is `nil`.

Example

```
techGetLayerProp( techFileID list("nwell" "drawing") "myProp")
=> "well"
```

Returns the `well` layer property value.

techGetTwoLayerProp

```
techGetTwoLayerProp(
     d_techFileID
     ltx_layer1
     ltx_layer2
     t_propName
     )
     => g_propValue/nil
```

Description

Returns the value of the specified two-layer property from the `techLayerProperties` subclass of the specified technology file. The `techLayerProperties` subclass is located in the Layer Definitions class; it specifies special properties that you want to place on the layers in your design.

For more information about the `techLayerProperties` subclass of the technology file, refer to "techLayerProperties" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file defining the layer.
ltx_layer1	The first layer. A list containing the layer name and purpose, or the layer name, or the layer number of the layer.
ltx_layer2	The second layer. A list containing the layer name and purpose, or the layer name, or the layer number of the layer.
t_propName	The name of the property.

Return Values

g_propValue	The value of the property.
`nil`	The technology file or layer does not exist or the property value is `nil`.

Example

```
techGetTwoLayerProp(techFileID "via" "metal1" "myProp")
=> "router"
```

Returns the value of the `myProp` property set on the `via` and `metal1` layers in the technology file identified by `techFileID`.

techDeleteTwoLayerProp

```
techDeleteTwoLayerProp(
     d_techFileID
     ltx_layer1
     ltx_layer2
     t_name
     )
     => t/nil
```

Description

Deletes the specified two-layer property from the `techLayerProperties` subclass of the specified technology file. The `techLayerProperties` subclass is located in the Layer Definitions class; it specifies special properties that you want to place on the layers in your design.

For more information about the `techLayerProperties` subclass of the technology file, refer to "techLayerProperties" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file defining the layer.
ltx_layer1	The first layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
ltx_layer2	The second layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
t_name	The name of the two-layer property to delete.

Return Values

`t`	The property was deleted.
`nil`	The technology file or the property does not exist.

Example

```
techDeleteTwoLayerProp( techFileID "metal" "via" "minSpacing" )
=> t
```

Deletes the `minSpacing` property between `metal` and `via` layers in the technology file identified by `techFileID`.

techSetLayerFunction

```
techSetLayerFunction(
     d_techFileId
     ltx_layer
     g_function
     )
     => t/nil
```

Description

Updates the function of the specified layer in the `layerFunctions` subclass of the specified technology file. The `layerFunctions` subclass is located in the Layer Rules class; it assigns functions to layers. If a `layerFunctions` subclass does not exist, this function creates one with the specified data.

Arguments

d_techFileId	The database identifier of the technology file.
ltx_layer	The layer name, layer number, or a list containing the layer name and purpose.
g_function	The layer function. Valid Values: `cut`, `li`, `metal`, `ndiff`, `nplus`, `nwell`, `pdiff`, `poly`, `pplus`, `pwell`

Return Values

`t`	The function was set.
`nil`	The technology file does not exist or the layer is not defined in the technology file.

Example

```
techSetLayerFunction( tfID "metal1" "metal" )
=> t
```

Assigns the layer function `metal` to the layer `metal1` in the technology file identified by `tfID`.

techSetLayerFunctions

```
techSetLayerFunctions(
     d_techFileId
     l_layerFunctionsList
     )
     => t/nil
```

Description

Updates the functions of the specified layers in the `layerFunctions` subclass of the specified technology file. The `layerFunctions` subclass is located in the Layer Rules class; it assigns functions to layers. If a `layerFunctions` subclass does not exist, this function creates one with the specified data.

Arguments

`d_techFileId` The database identifier of the technology file.

`l_layerFunctionsList`
A list specifying the layer names and functions to assign to them. The list has the following syntax:

```
( ( ltx_layer g_function )... )
```

`ltx_layer` is the layer name, layer number, or a list containing the layer name and purpose

`g_function` is the layer function
Valid Values: `cut`, `li`, `metal`, `ndiff`, `nplus`, `nwell`, `pdiff`, `poly`, `pplus`, `pwell`

Return Values

`t` The functions are assigned to the layers.

`nil` The technology file does not exist, the layer is not defined in the technology file, or the function is not a valid function.

Example

```
techSetLayerFunctions( tfID '( ("poly2" "poly") ("metal3" "metal") ) )
=> t
```

Assigns the layer function `poly` to the layer `poly2` and the layer function `metal` to the layer `metal3` in the technology file identified by `tfID`.

techGetLayerFunction

```
techGetLayerFunction(
     d_techFileId
     ltx_layer
     )
     => g_function/nil
```

Description

Returns the function assigned to the specified layer.

Arguments

d_techFileId	The database identifier of the technology file.
ltx_layer	The layer name, layer number, or a list containing the layer name and purpose.

Return Values

g_function	The layer function.
nil	The technology file does not exist or the layer is not defined in the technology file.

Example

```
techGetLayerFunction( tfID "metal1" )
=> "metal"
```

Returns the function (`metal`) assigned to the layer `metal1` in the technology file identified by `tfID`.

techGetLayerFunctions

```
techGetLayerFunctions(
     d_techFileId
     )
     => l_layerFunctions/nil
```

Description

Returns a list of the functions assigned to the specified layer.

Arguments

d_techFileId	The database identifier of the technology file.

Return Values

l_layerFunctions	List of the functions assigned to a specified layer.
nil	The technology file does not exist.

Example

```
techGetLayerFunctions( tfID )
=> (("metal1" "metal")
       ("pwell" "pwell")
       ("via" "cut")
       ("poly1" "poly")
)
```

Returns the layers with functions assigned to them, along with their functions, in the technology file identified by `tfID`. These are layer `metal1` with function `metal`, layer `pwell` with function `pwell`, layer `via` with function `cut`, and layer `poly1` with function `poly`.

3

Device SKILL Functions

Symbolic layout is a method of creating custom integrated circuit layouts using symbolic devices instead of polygons. The Devices class defines basic symbolic devices such as transistors, contacts, and pins.

Symbolic devices are similar to parameterized cells except that symbolic devices can be defined only in the technology file. In addition, symbolic devices have two levels of parameterization: class parameters and formal parameters. Cadence provides a set of predefined device types. You can also define class and formal parameters to create your own device types.

techGetDeviceCellView

```
techGetDeviceCellView(
     d_techFileID
     t_deviceName
     t_viewName
     )
     => d_cellViewID/nil
```

Description

Loads the supermaster cellview of the specified device in virtual memory and returns the associated database identifier. The cellview is open in read mode. You can use `dbClose` to close the cellview. The cellview must be defined as a device in the Devices class of the specified technology file.

For more information about the Devices class, refer to "Devices Class" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file that defines the device.
t_deviceName	The name of the device.
t_viewName	The view name of the device. Valid Values: `symbolic`

Return Values

d_cellViewID	The database identifier of the supermaster of the device.
`nil`	The technology file the cellview is not defined in the technology file.

Example

```
techGetDeviceCellView(techFileID "NTR" "symbolic")
=> db:20953132
```

Opens the `symbolic` view of the `NTR` cell and returns the database identifier `20953132`.

techGetDeviceCParam

```
techGetDeviceCParam(
     d_techFileID
     t_deviceName
     t_viewName
     )
     => l_paramValue/nil
```

Description

Returns a list of the names and values of the class parameters of the specified device. Devices are defined in the Devices class of the technology file.

For more information about the Devices class, refer to "Devices Class" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file defining the device.
t_deviceName	The name of the device.
t_viewName	The view name of the device. Valid Values: `symbolic`

Return Values

l_paramValue	A list of class parameter name and value pairs. The list has the following syntax: ((*t_paramName g_paramValue*) ...) *t_paramName* is the name of the class parameter. *g_paramValue* is the value assigned to the parameter when the device was created.
`nil`	The technology file does not exist or the device is not defined.

Example

```
techGetDeviceCParam( techFileID "NTR" "symbolic" )
=> ((sdImpEnc 0.0)
(("sdImpLayer" nil)
("gateExt" 1.000000)
("sdExt" 2.600000)
("gateLayer" "POLY")
("sdLayer" "S_D")
)
```

Returns the class parameters defined for the device named `NTR` in the technology file identified by `techFileID`. The parameters are `sdImpEnc`, `sdImpLayer`, `gateExt`, `sdExt`, `gateLayer`, and `sdLayer`.

techGetDeviceFParam

```
techGetDeviceFParam(
     d_techFileID
     t_deviceName
     t_viewName
     )
     => l_paramValue/nil
```

Description

Returns a list of the names and values of the formal parameters defined of the specified device. Devices are defined in the Devices class of the technology file.

For more information about the Devices class, refer to "Devices Class" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file defining the device.
t_deviceName	The name of the device.
t_viewName	The view name on which the device is defined. Valid Values: `symbolic`

Return Values

l_paramValue	A list of formal parameter name and default pairs. The list has the following syntax: ((*t_paramName g_paramValue*) ...) *t_paramName* is the name of the formal parameter. *g_paramValue* is the value assigned to the parameter when the device was created.
`nil`	The technology file does not exist or the device is not defined.

Example

```
techGetDeviceFParam( techFileID "NTR" "symbolic" )
=> (("l" 2.000000)("w" 3.600000))
```

Returns the formal parameters and values for the `NTR` device defined in the technology file identified by `techFileID`.

techGetDeviceInClass

```
techGetDeviceInClass(
     d_techFileID
     t_deviceType
     t_viewName
     )
     => l_devices/nil
```

Description

Returns a list of the names of all devices created in the specified technology file, device type, and view. Devices are defined in the Devices class of the technology file.

For more information about the Devices class, refer to "Devices Class" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file defining the device.
t_deviceType	The device type name.
t_viewName	The view name on which the device is defined. Valid Values: `symbolic`

Return Values

l_devices	A list of device names defined in the specified device type.
`nil`	The technology file does not exist, or no devices of the specified device type are defined.

Example

```
techGetDeviceInClass( techFileID "syEnhancement" "symbolic" )
=> ("NTR" "PTR")
```

Returns the `NTR` and `PTR` device names, which are `syEnhancement` devices defined in the technology file identified by `techFileID`.

techGetDeviceClassViewList

```
techGetDeviceClassViewList(
     d_techFileID
     )
     => nil/l_deviceClassViews
```

Description

Returns the view name of a particular list of classes or all classes if `nil/ no arg` is supplied.

For more information about the Devices class, refer to "Devices Class" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the instance you want to check.

Return Values

`nil`	The technology file does not exist, or no devices are defined for the device type.
l_deviceClassViews	The view name of a particular list of classes or all classes if `nil/ no arg` is supplied.

Example

```
tf=techGetTechFile(ddGetObj("tlib"))
techGetDeviceClassViewList(tf) => ("symbolic")
```

Returns the view name `symbolic` for the device classes in the technology file `tlib`.

techIsDevice

```
techIsDevice(
     d_instID
     )
     => t/nil
```

Description

Indicates whether the supermaster of the specified instance is defined as a device in the Devices class of the technology file.

For more information about the Devices class, refer to "Devices Class" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_instID	The database identifier of the instance you want to check.

Return Values

t	The specified cellview is a device.
nil	The technology file does not exist or the cellview is not defined as a device.

Example

```
techIsDevice( techFileID cellID )
=> t
```

The cellview identified by `cellID` is defined as a device in the technology file identified by `techFileID`.

techSetDeviceProp

```
techSetDeviceProp(
     d_techFileID
     t_deviceName
     t_viewName
     l_propertyValue
     )
     => t/nil
```

Description

Updates the value of the specified device property in the `deviceProps` section of the specified technology file. A `deviceProps` section follows each of the device type definition sections in the Devices class. If the property does not exist, this function creates it.

For more information about the Devices class, refer to "Devices Class" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	Database identifier of the technology file defining the device.
t_deviceName	The device name.
t_viewName	The view name on which the device is defined. Valid Values: `symbolic`
l_propertyValue	A list of the property name and value. The list has the following syntax: `( t_propName g_value )` *t_propName* is the name of property to set or create. *g_value* is the value of the property. Valid Values: an integer, a floating-point number, a string enclosed in quotes, a Boolean value, any SKILL symbol or expression that evaluates to any of these types

Return Values

`t`	The device property was set or created.

`nil`	The device does not exist.

Example

```
techSetDeviceProp( techFileID "syDepletion" "symbolic" list("function"
"transistor") )
```

Sets the `function` property to `transistor` for the `syDepletion` device in the technology file identified by `techFileID`.

techGetDeviceProp

```
techGetDeviceProp(
     d_techFileID
     t_deviceName
     t_viewName
     t_propName
     )
     => g_value/nil
```

Description

Returns the value of the specified device property from the `deviceProps` section of the specified technology file. A `deviceProps` section follows each of the device type definition sections in the Devices class. If the property is not found in the `deviceProps` section, the system looks in the `deviceClassProps` section for the specified property on the device class of the specified device.

For more information about the Devices class, refer to "Devices Class" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file defining the device.
t_deviceName	The device name.
t_viewName	The view name on which the device is defined. Valid Values: `symbolic`
t_propName	The property name.

Return Values

g_value	Value of the property.
`nil`	The technology file, device, or property does not exist or the property value is `nil`.

Example

```
techGetDeviceProp( techFileID "syContact" "symbolic" "function")
=> "ptr"
```

Returns the value of the `function` property defined for the `syContact symbolic` device in the technology file identified by `techFileID`.

tfcDeleteDeviceProp

```
tfcDeleteDeviceProp(
     t_viewName
     t_deviceName
     t_propName
     g_propValue
     )
     => t/nil
```

Description

Deletes device properties.

Arguments

t_viewName	The name of the view from which you want to delete device properties.
t_deviceName	The name of the device from which you want to delete device properties.
t_propName	The name of the property you want to delete.
t_propValue	The value assigned to the property.

Return Values

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

techGetDeviceClass

```
techGetDeviceClass(
     d_techFileID
     [t_deviceName]
     t_viewName
     )
     => l_types/t_type/nil
```

Description

Returns a list of the names of all device types defined with the specified view in the current technology file. Device types and devices are defined in the Devices class of the technology file. If you specify a device name, this function returns the device type of the device.

For more information about the Devices class, refer to "Devices Class" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file defining the device type.
t_deviceName	The device name. Valid Values: name of an existing device, `nil`
t_viewName	The view name defined for the device type. Valid Values: `symbolic`

Return Values

l_types	A list of device type names of all device types defined with the specified view in the current technology file. If you specify a device name, returns one device type name.
t_type	The device type name of the device specified by `deviceName`.
`nil`	The technology file or device does not exist.

Examples

```
techGetDeviceClass( techFileID "symbolic" )
=> ("syEnhancement" "syPin")
```

Returns a list of all of the device types defined for the `symbolic` view.

```
techGetDeviceClass( techFileID "NTR" "symbolic" )
=> "syEnhancement"
```

Returns the device type of the `NTR symbolic` device.

techSetDeviceClassProp

```
techSetDeviceClassProp(
     d_techFileID
     t_deviceType
     t_viewName
     l_propertyValue
     )
     => t/nil
```

Description

Updates the value of the specified device type property. Device type properties are defined in the Devices class of the technology file. If the property does not exist, this function creates it.

For more information about the Devices class, refer to "Devices Class" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	Database identifier of the technology file defining the device type.
t_deviceType	The device type name.
t_viewName	The view name for the device type. Valid Values: `symbolic`
l_propertyValue	A list of the property name and value. The list has the following syntax: `( t_propName g_value )` *t_propName* is the name of the property to set or create. *g_value* is the value of the property. Valid Values: an integer, a floating-point number, a string enclosed in quotes, a Boolean value, any SKILL symbol or expression that evaluates to any of these types

Return Values

`t`	The device type property was set or created.
`nil`	The technology file or device type does not exist.

Example

```
techSetDeviceClassProp( techFileID " "leTran" "symbolic list("function"
"transistor") )
```

Sets the `function` property of the device type `leTran`, view name `symbolic` to `transistor` in the technology file identified by `techFileID`.

techGetDeviceClassProp

```
techGetDeviceClassProp(
     d_techFileID
     t_deviceType
     t_viewName
     t_propName
     )
     => g_propValue/nil
```

Description

Returns the value of the specified device type property. Device type properties are defined in the Devices class of the technology file.

For more information about the Devices class, refer to "Devices Class" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_deviceType	The device type name.
t_viewName	The view name on which the device type is defined. Valid Values: `symbolic`
t_propName	The property name.

Return Values

g_propValue	The value of the property. Valid Values: an integer, a floating-point number, a string enclosed in quotes, a Boolean value, any SKILL symbol that evaluates to any of these types
`nil`	The technology file does not exist, the device type property is not defined, or the property value is `nil`.

Example

```
techGetDeviceClassProp(techFileID "syEnhancement" "symbolic" "function" )
=> "transistor"
```

Returns the value of the `function` property on the `syEnhancement` device type defined for the `symbolic` view in the technology file identified by `techFileID`.

techDeleteDeviceClass

```
techDeleteDeviceClass(
     d_techFileID
     t_viewName
     t_className
     )
     => t/nil
```

Description

Deletes a device of a defined device type previously created using `tcCreateDeviceClass` and `tcDeclareDevice`.

For more information about the Devices class, refer to "Devices Class" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_viewName	The name of the view for the device you want to delete.
t_className	The class of the device you want to delete.

Return Values

`t`	The device was deleted.
`nil`	The technology file or device class does not exist. **Note:** The software displays a warning message along with returning `nil`.

Example

Assume that your technology file contains the following data creating a device:

```
devices(
         tcCreateDeviceClass( "symbolic" "eg1"

; class parameter name value pairs
   ()

; formal parameter name value pairs
   (
```

```
     (xLen 2.4)
     (yLen 1.2)
)

; access controls techfile class for layout design rules
; and layer info

; set origin reference coordinates
   oriX = 0
   oriY = 0

; cvId drawing layer
   dbCreateRect(tcCellView list("pdiff" "drawing")

; lower left and upper right coords of rectangle
   list( oriX:oriY xLen:yLen)
)

)
; tcCreateDeviceClass

; declare a cv of device class eg1 called eg1
        tcDeclareDevice( "symbolic" "eg1" "eg1")
)
```

The following deletes the device created with this technology data:

```
techDeleteDeviceClass(techGetTechFile(ddGetObj("exempliGratia") "symbolic" "eg1")
```

4

Physical and Electrical Rules SKILL Functions

This chapter describes:

- Physical Rules SKILL Functions
- Electrical Rules SKILL Functions

Physical Rules SKILL Functions

The Physical Rules class lets you define the following:

- Spacing information for single layers (for example, width and notch spacing rules)
- Spacing information for two layer (for example, the minimum distance allowed between objects on the same layer or different layers)
- The amount of space required when one object encloses another
- The manufacturing grid resolution, defining when grid snapping must be a multiple of a specified value

techSetSpacingRule

```
techSetSpacingRule(
     d_techFileID
     t_rule
     g_value
     ltx_layer1
     [ltx_layer2]
     )
     => t/nil
```

Description

Updates the value of the spacing rule for the specified layer or layers in the `spacingRules` subclass of the technology file. The `spacingRules` subclass is located in the Physical Rules class; it defines the spacing requirements of your design environment. If you do not specify a second layer, the spacing rule applies only to the first layer. If a spacing rule does not exist for the specified layer or layers, the function creates one. If a `spacingRules` subclass does not exist, this function creates one with the specified data.

For more information about the `spacingRules` subclass of the technology file, refer to "spacingRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_rule	The name of the spacing rule. Valid Values: any string (examples: `minNotch`, `minSpacing`, `minWidth`, `defaultWidth`, `maxWidth`, `sameNet`)
g_value	The value of the spacing rule. Valid Values: any floating-point number, any integer
ltx_layer1	The first layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
ltx_layer2	The optional second layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

`t`	The spacing rule was updated or created successfully.
`nil`	The technology file does not exist or the layers are not defined.

Example

```
techSetSpacingRule( techFileID "minSpacing" 0.6 "metal1" )
=> t
```

Sets the minimum spacing rule for `metal1` objects to `0.6` user units in the technology file identified by `techFileID`.

techGetSpacingRules

```
techGetSpacingRules(
     d_techFileID
     )
     => l_spacingRules/nil
```

Description

Returns a list of the spacing rules defined in the `spacingRules` subclass of the technology file. The `spacingRules` subclass is located in the Physical Rules class; it defines the spacing requirements of your design environment.

For more information about the `spacingRules` subclass of the technology file, refer to "spacingRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_spacingRules	A list of lists containing the spacing rule settings. The list has the following syntax: `( ( t_rule g_value lt_layer1 [lt_layer2] ) ... )` *t_rule* is the name of the spacing rule. *g_value* is the value of the spacing rule. *lt_layer1* is the first layer. *lt_layer2* is the second layer. Returned only for two-layer spacing rules.
`nil`	The technology file does not exist or there are no spacing rules defined.

Example

```
techGetSpacingRules( techFileID )
=> (( minWidth 0.60 cont )
    ( minSpacing 0.60 cont )
    ( minSpacing 0.60 cont poly1 )
    ( minSpacing 0.60 cont via ))
```

Returns the spacing rules defined in the technology file identified by `techFileID`.

techGetSpacingRule

```
techGetSpacingRule(
     d_techFileID
     t_rule
     ltx_layer1
     [ltx_layer2]
     )
     => g_value/nil
```

Description

Returns the value of the specified spacing rule defined in the `spacingRules` subclass of the technology file. The `spacingRules` subclass is located in the Physical Rules class; it defines the spacing requirements of your design environment.

For more information about the `spacingRules` subclass of the technology file, refer to "spacingRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_rule	The name of the spacing rule.
ltx_layer1	The first layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
ltx_layer2	The optional second layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

g_value	The value of the spacing rule.
`nil`	The technology file does not exist or there are no spacing rules defined.

Example

```
techGetSpacingRule( techFileID "minWidth" "metal1" )
=> 0.6
```

Returns the value of the `minWidth` spacing rule defined for the `metal1` layer in the technology file identified by `techFileID`.

techSetOrderedSpacingRule

```
techSetOrderedSpacingRule(
     d_techFileID
     t_rule
     g_value
     ltx_layer1
     ltx_layer2
     )
     => t/nil
```

Description

Updates the value of the spacing rule for the specified layers in the `orderedSpacingRules` subclass of the technology file. The `orderedSpacingRules` subclass is located in the Physical Rules class; it defines the spacing requirements for which the order of the layers is important. If an ordered spacing rule does not exist for the specified layers, the function creates one. If an `orderedSpacingRules` subclass does not exist, this function creates one with the specified data.

For more information about the `orderedSpacingRules` subclass of the technology file, refer to "orderedSpacingRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_rule	The name of the spacing rule. Valid Values: any string (example: `minEnclosure`)
g_value	The value of the spacing rule. Valid Values: any floating-point number, any integer
ltx_layer1	The first layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
ltx_layer2	The second layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

`t`	The spacing rule was updated or created successfully.
`nil`	The technology file does not exist or the layers are not defined.

Example

```
techSetOrderedSpacingRule( techFileID "minEnclosure" 0.6 "via" "metal1" )
=> t
```

Sets the minimum enclosure rule for the `via` and `metal1` layers to `0.6` user units in the technology file identified by `techFileID`.

Note that this rule applies to all layer-purpose pairs with the layer names `via` and `metal1`. If you want a different spacing for a specific layer-purpose pair with either of these names, you must list it separately. For example:

```
techSetOrderedSpacingRule( techFileID "minEnclosure" 0.4 "via" "pin" "metal1"
"pin" )
```

techGetOrderedSpacingRules

```
techGetOrderedSpacingRules(
     d_techFileID
     )
     => l_spacingRules/nil
```

Description

Returns a list of all of the ordered spacing rules defined in the `orderedSpacingRules` subclass of the technology file. The `orderedSpacingRules` subclass is located in the Physical Rules class; it defines the spacing requirements for which the order of the layers is important.

For more information about the `orderedSpacingRules` subclass of the technology file, refer to "orderedSpacingRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_spacingRules	A list of lists containing the ordered spacing rule settings. The list has the following syntax: `( ( t_rule g_value lt_layer1 lt_layer2 ) ... )` *t_rule* is the name of the spacing rule. *g_value* is the value of the spacing rule. *lt_layer1* is the first layer. *lt_layer2* is the second layer.
`nil`	The technology file does not exist or there are no spacing rules defined.

Example

```
techGetOrderedSpacingRules( techFileID )
=> (( minEnclosure  0.600000  metal1      via )
    ( minEnclosure  0.300000  pimplant    diff )
    ( minEnclosure  0.300000  prBoundary  cont )
    ( minEnclosure  0.300000  prBoundary  diff )
    ( minEnclosure  0.600000  prBoundary  metal1 ))
```

Returns the ordered spacing rules defined in the technology file identified by `techFileID`.

techGetOrderedSpacingRule

```
techGetOrderedSpacingRule(
     d_techFileID
     t_rule
     ltx_layer1
     ltx_layer2
     )
     => g_value/nil
```

Description

Returns the value of the specified ordered spacing rule defined in the `orderedSpacingRules` subclass of technology file. The `orderedSpacingRules` subclass is located in the Physical Rules class; it defines the spacing requirements for which the order of the layers is important.

For more information about the `orderedSpacingRules` subclass of the technology file, refer to "orderedSpacingRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_rule	The name of the ordered spacing rule.
ltx_layer1	The first layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
ltx_layer2	The second layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

g_value	The value of the ordered spacing rule.
`nil`	The technology file does not exist or the specified rule does not exist.

Example

```
techGetOrderedSpacingRule( techFileID "minEnclosure" "via" "metal1")
=> 0.6
```

Returns the value of the `minEnclosure` spacing rule defined for the `via` and `metal1` layers in the technology file identified by `techFileID`.

techCreateSpacingRuleTable

```
techCreateSpacingRuleTable(
     d_techFileId
     t_ruleName
     l_indexDefinitions
     ltx_layer1
     [ltx_layer2]
     )
     => t/nil
```

Description

Creates a spacing rules table and indices in the `tableSpacingRules` subclass of the technology file. The `tableSpacingRules` subclass is located in the Physical Rules class; it defines lookup tables used to determine spacing rules. This SKILL function does not create the table entries; to create table entries, use techSetSpacingRuleTableEntry. If a `tableSpacingRules` subclass does not exist, this function creates one.

Arguments

d_techFileId	The database identifier of the technology file.
t_ruleName	The name of the spacing rule.
l_indexDefinitions	A list defining the name, predefined index values, and user-defined match function for the table. The list has the following syntax: `( [`*t_index1Name*`] [`*l_def1Indices*`] [`*t_match1Func*`] [`*t_index2Name*`] [`*l_def2Indices*`] [`*t_match2Func*`] )` *t_index1Name* is the name of the first dimension of a two-dimensional table or the only dimension of a one-dimensional table. Valid Values: Any number or string *l_def1Indices* is a list of predefined indices for the first dimension, specifying the indices that can be used in the table. *t_match1Func* is the name of a user-defined match function. Valid Values: <, <=, =, >, >=, *t_userMatchFunction*, `nil` where *userMatchFunction* is a user-defined function.

Default: ==

t_index2Name is the name of the second dimension of a two-dimensional table.
Valid Values: Any number or string

l_def2Indices is a list of predefined indices for the second dimension, specifying the indices that can be used in the table.

t_match2Func is the name of a user-defined match function.
Valid Values: <, <=, =, >, >=, *t_userMatchFunction*, nil where *userMatchFunction* is a user-defined function.
Default: ==

ltx_layer1 — The first layer on which to apply the table rule.
Valid Values: the layer name, the layer number, a list containing the layer name and purpose

ltx_layer2 — The optional second layer on which to apply the table rule.
Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

t — The spacing rule table was created.

nil — The spacing rule table was not created.

Example

```
techCreateSpacingRuleTable( tfID "WIDTHS" '("minWidth" "maxWidth") "metal1" )
=> t
```

Creates the table spacing rule WIDTHS with the indices minWidth and maxWdth applied to layer metal1.

techGetSpacingRuleTable

```
techGetSpacingRuleTable(
     d_techFileId
     t_rule
     ltx_layer1
     [ltx_layer2]
     )
     => l_tables/nil
```

Description

Returns the table data for the specified table spacing rule defined in the `tableSpacingRules` subclass of the technology file and applied to the specified layer or layers. The `tableSpacingRules` subclass is located in the Physical Rules class; it defines lookup tables used to determine spacing rules.

Arguments

d_techFileId	The database identifier of the technology file.
t_rule	The name of the spacing rule.
ltx_layer1	The first layer on which to apply the table rule. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
ltx_layer2	The optional second layer on which to apply the table rule. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

l_tables	A list defining the table entries. The list has the following syntax: `(( l_index1Definitions [l_index2Definitions]) l_table )` *l_index1Definitions* is a list defining the name, predefined index values, and user-defined match function for the first table dimension. The format of the list is as follows:

```
[nt_indexName] [( lg_indexValue... )]
[t_userMatchFunction]
```

l_index2Definitions is a list defining the name, predefined index values, and user-defined match function for the first table dimension. The list has the following syntax:

```
[nt_indexName] [( lg_indexValue... )]
[t_userMatchFunction]
```

l_table is a list of the table entries.

`nil` The technology file does not exist or there are no spacing rule tables defined.

Example

```
techGetSpacingRuleTable( tfID "MIN" "via" )
=> (("minWidth"
        (1000000.0 1e+08) nil)
         1000000.0 5e-07 1e+08 4e-07
      )
```

Returns the table data for the `MIN` rule applied to `via` in the technology file identified by `tfID`. There is a single index (with index name `minWidth`) and the table contains two index entries: `1000000.0` (with associated value `5e-07`) and `1e+08` (with associated value `4e-07`).

techGetSpacingRuleTables

```
techGetSpacingRuleTables(
     d_techFileId
     )
     => l_tables/nil
```

Description

Returns a list of the spacing rule tables and the layers to which they apply as defined in the `tableSpacingRules` subclass of the technology file. The `tableSpacingRules` subclass is located in the Physical Rules class; it defines lookup tables used to determine spacing rules.

Arguments

d_techFileId	The database identifier of the technology file.

Return Values

l_tables	A list of the table spacing rule names and the layers to which they apply. The list has the following syntax: `( ( t_rule ltx_layer1 [ltx_layer2] ) ... )` *t_rule* is the name of the spacing rule. *ltx_layer1* is the first layer on which the table rule is applied. *ltx_layer2* is the optional second layer on which the table rule is applied.
`nil`	The technology file does not exist or there are no spacing rules tables defined.

Example

```
techGetSpacingRuleTables( tfID )
=> (("MIN" "via")
    ("MAX" "via")
    ("DEFAULT" "via")
    ("MIN RULES" "metal1")
)
```

Returns the names of the table spacing rules and the layers to which each applies in the technology file identified by `tfID`. These are all single-layer spacing rules.

techSetSpacingRuleTableEntry

```
techSetSpacingRuleTableEntry(
     d_techFileId
     t_rule
     l_index
     g_value
     ltx_layer1
     [ltx_layer2]
     )
     => t/nil
```

Description

Sets the value of the specified index in the specified spacing rule table defined in the `tableSpacingRules` subclass of the technology file. The `tableSpacingRules` subclass is located in the Physical Rules class; it defines lookup tables used to determine spacing rules.

Arguments

d_techFileId	The database identifier of the technology file.
t_rule	The name of the spacing rule.
l_index	The list of indices. The list has the following syntax: `( g_index1 [g_index2] )` *g_index1* is the first index in a two-dimensional table or the only index in a one-dimensional table. *g_index2* is the second index in a two-dimensional table.
g_value	The value to use when the application finds a match to the index or pair of indices during table lookup. Valid Values: Any number or string
ltx_layer1	The first layer on which the table rule is applied. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

ltx_layer2	The optional second layer on which the table rule is applied. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

t	The value was successfully set.
nil	The technology file does not exist, the table spacing rule is not defined, or the index is not defined.

Example

```
techSetSpacingRuleTableEntry( tfID "MIN" 0 "5.0" "via" )
=> t
```

Sets the value of the index `0` for the `MIN` rule applied to `via` in the technology file identified by `tfID` to `5.0`.

techGetSpacingRuleTableEntry

```
techGetSpacingRuleTableEntry(
     d_techFileId
     t_rule
     l_index
     ltx_layer1
     [ltx_layer2]
     )
     => g_value/nil
```

Description

Returns the value of the specified index in the specified spacing rule table defined in the `tableSpacingRules` subclass of the technology file. The `tableSpacingRules` subclass is located in the Physical Rules class; it defines lookup tables used to determine spacing rules.

Arguments

d_techFileId	The database identifier of the technology file.
t_rule	The name of the spacing rule.
l_index	The list of indices. The list has the following syntax: `( g_index1 [g_index2] )` *g_index1* is the first index in a two-dimensional table or the only index in a one-dimensional table. *g_index2* is the second index in a two-dimensional table.
ltx_layer1	The first layer on which the table rule is applied. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
ltx_layer2	The optional second layer on which the table rule is applied. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

g_value	The value of the specified index.
nil	The technology file does not exist or the table spacing rule is not defined.

Example

```
techGetSpacingRuleTableEntry( tfID "MIN" 0 "via" )
=> 5.5
```

Returns the value (5.5) of the index 0 for the table rule MIN applied to the layer via in the technology file identified by tfID.

techSetMfgGridResolution

```
techSetMfgGridResolution(
     d_techFileID
     g_resolution
     )
     => t/nil
```

Description

Updates or sets the value used for grid snapping in the `mfgGridResolution` subclass of the technology file. The `mfgGridResolution` subclass is located in the Physical Rules class. When specified, it establishes that grid snapping must be a multiple of the value of *`g_resolution`*. The Virtuoso Compactor uses this value when compacting designs.

For more information about the `mfgGridResolution` subclass of the technology file, refer to "mfgGridResolution" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`g_resolution`	The number of user units to use as the basis for the grid. Valid Values: any floating-point number, any integer

Return Values

`t`	The manufacturing grid resolution was updated or created successfully.
`nil`	The technology file does not exist.

Example

```
techSetMfgGridResolution ( techFileID 0.002000 )
=> t
```

Sets the manufacturing grid resolution value to `0.002000` in the technology file identified by `techFileID`.

techGetMfgGridResolution

```
techGetMfgGridResolution(
     d_techFileID
     )
     => g_resolution/nil
```

Description

Returns the value of the manufacturing grid resolution defined in the `mfgGridResolution` subclass of technology file. The `mfgGridResolution` subclass is located in the Physical Rules class. When specified, it establishes that grid snapping must be a multiple of the value of *`g_resolution`*. The Virtuoso Compactor uses this value when compacting designs.

For more information about the `mfgGridResolution` subclass of the technology file, refer to "mfgGridResolution" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.

Return Values

`g_resolution`	The number of user units to use as the basis for the grid, as specified in the `mfgGridResolution` subclass of the specified technology file.
`nil`	The technology file does not exist or no manufacturing grid resolution is specified.

Example

```
techGetMfgGridResolution ( techFileID )
=> 0.002000
```

Returns the value `0.002000`, which is the manufacturing grid resolution specified in the technology file identified by `techFileID`.

Electrical Rules SKILL Functions

The Electrical Rules class lets you define the electrical characteristics of a layer (for example, capacitance, density, resistance, area capacitance, and so on).

techSetElectricalRule

```
techSetElectricalRule(
     d_techFileID
     t_rule
     g_value
     ltx_layer1
     [ltx_layer2]
     )
     => t/nil
```

Description

Updates the value of the electrical rule for the specified layer or layers in the `characterizationRules` subclass of the technology file Electrical Rules class. The `characterizationRules` subclass specifies the electrical properties of the layers you use in your designs. If you do not specify a second layer, the rule applies only to the first layer. If a characterization rule does not exist for the specified layer or layers, the function creates one. If a `characterizationRules` subclass does not exist, this function creates one with the specified data.

For more information about the `characterizationRules` subclass of the technology file, refer to "characterizationRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_rule	The name of the characterization rule. Valid Values: any string (examples: `areaCap`, `currentDensity`, `edgeCap` `sheetRes`, `height`, `thickness`, `shrinkage`, `capMultiplier`, `parallelCap`)
g_value	The value of the characterization rule. Valid Values: any floating-point number, any integer
ltx_layer1	The first layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
ltx_layer2	The optional second layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

`t`	The characterization rule was updated or created successfully.
`nil`	The technology file does not exist or the layers are not defined.

Example

```
techSetElectricalRule( techFileID "areaCap" 0.6 "metal1")
=> t
```

Sets the area capacitance rule for `metal1` objects to `0.6` user units in the technology file identified by `techFileID`.

techGetElectricalRules

```
techGetElectricalRules(
     d_techFileID
     )
     => l_electricalRules/nil
```

Description

Returns a list of all of the electrical rules defined in the `characterizationRules` subclass of the technology file. The `characterizationRules` subclass is located in the Electrical Rules class; it specifies the electrical properties of the layers you use in your designs.

For more information about the `characterizationRules` subclass of the technology file, refer to "characterizationRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_electricalRules	A list of lists containing the characterization rule settings. The list has the following syntax: `( ( t_rule g_value lt_layer1 [lt_layer2] ) ... )` *t_rule* is the name of the characterization rule. *g_value* is the value of the characterization rule. *lt_layer1* is the first layer. *lt_layer2* is the second layer. Only returned for two-layer characterization rules.
`nil`	The technology file does not exist or there are no electrical characterization rules defined.

Example

```
techGetElectricalRules( techFileID )
=>(( areaCap 1.4e-4 "metal1" )
   ( currentDensity 2.000000 "metal1" )
   ( edgeCapacitance 4.0e-11 "metal1" )
   ( sheetRes 0.040000 "metal1" ))
```

Returns the characterization rules defined in the technology file identified by `techFileID`.

techGetElectricalRule

```
techGetElectricalRule(
     d_techFileID
     t_rule
     ltx_layer1
     [ltx_layer2]
     )
     => g_value/nil
```

Description

Returns the value of the specified electrical rule defined in the specified technology file. The `characterizationRules` subclass is located in the Electrical Rules class; it specifies the electrical characteristics of the layers you use in your designs.

For more information about the `characterizationRules` subclass of the technology file, refer to "characterizationRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_rule	The name of the ordered characterization rule.
ltx_layer1	The first layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
ltx_layer2	The optional second layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

g_value	The value of the characterization rule.
`nil`	The technology file does not exist, the specified characterization rule is not defined for the specified layer or layers, or the specified layer or layers do not exist.

Example

```
techGetElectricalRule( techFileID "minWidth" "metal1" )
=> 0.6
```

Returns the `minWidth` characterization rule defined for the `metal1` layer in the technology file identified by `techFileID`.

techSetOrderedElectricalRule

```
techSetOrderedElectricalRule(
     d_techFileID
     t_rule
     g_value
     ltx_layer1
     ltx_layer2
     )
     => t/nil
```

Description

Updates the value of the electrical rule for the specified layers in the `orderedCharacterizationRules` subclass of the technology file. The `orderedCharacterizationRules` subclass is located in the Electrical Rules class; it specifies the electrical characteristics for which the order of the layers is important. If a rule does not exist for the specified layer or layers, the function creates one. If an `orderedCharacterizationRules` subclass does not exist, this function creates one with the specified data.

For more information about the `orderedCharacterizationRules` subclass of the technology file, refer to "orderedCharacterizationRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_rule	The name of the ordered characterization rule. Valid Values: any string
g_value	The value of the electrical rule. Valid Values: any floating-point number, any integer
ltx_layer1	The first layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
ltx_layer2	The second layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

`t`	The ordered characterization rule was updated or created successfully.
`nil`	The technology file does not exist or the layers are not defined.

Example

```
techSetOrderedElectricalRule( techFileID "minEnclosure" 0.6 "via" "metal1" )
=> t
```

Sets the minimum enclosure rule for the `via` and `metal1` layers to `0.6` user units in the technology file identified by `techFileID`.

techGetOrderedElectricalRules

```
techGetOrderedElectricalRules(
     d_techFileID
     )
     => l_electricalRules/nil
```

Description

Returns a list of all of the rules defined in the `orderedCharacterizationRules` subclass of the technology file. The `orderedCharacterizationRules` subclass is located in the Electrical Rules class; it specifies the electrical characteristics for which the order of the layers is important.

For more information about the `orderedCharacterizationRules` subclass of the technology file, refer to "orderedCharacterizationRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_electricalRules	A list of lists containing the ordered electrical rule settings. The list has the following syntax: `( ( t_rule g_value lt_layer1 lt_layer2 ) ... )` *t_rule* is the name of the ordered characterization rule. *g_value* is the value of the ordered characterization rule. *lt_layer1* is the first layer. *lt_layer2* is the second layer.
`nil`	The technology file does not exist or there are no ordered characterization rules defined.

Example

```
techGetOrderedElectricalRules( techFileID )
=> ( parallelCap 2.00 metal1  metal2 )
   ( parallelCap 2.00 metal3  metal4 )
```

Returns the ordered characterization rules defined in the technology file identified by `techFileID`.

techGetOrderedElectricalRule

```
techGetOrderedElectricalRule(
     d_techFileID
     t_rule
     ltx_layer1
     ltx_layer2
     )
     => g_value/nil
```

Description

Returns the value of the specified ordered characterization rule defined in the `orderedCharacterizationRules` subclass of the technology file. The `orderedCharacterizationRules` subclass is located in the Electrical Rules class; it specifies the electrical properties in which the order of the layers is important.

For more information about the `orderedCharacterizationRules` subclass of the technology file, refer to "orderedCharacterizationRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_rule	The name of the ordered electrical rule.
ltx_layer1	The first layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
ltx_layer2	The second layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

g_value	The value of the ordered characterization rule.
`nil`	The technology file does not exist, the specified rule does not exist for the specified layers, or the layers do not exist.

Example

```
techGetOrderedElectricalRule( techFileID "minEnclosure" "via" "metal1" )
=> 0.6
```

Returns the `minEnclosure` ordered characterization rule defined for the `via` and `metal1` layers in the technology file identified by `techFileID`.

techCreateElectricalRuleTable

```
techCreateElectricalRuleTable(
     d_techFileId
     t_ruleName
     l_indexDefinitions
     ltx_layer1
     [ltx_layer2]
     )
     => t/nil
```

Description

Creates an electrical characterization rules table and indices in the `tableCharacterizationRules` subclass of the technology file. The `tableCharacterizationRules` subclass is located in the Electrical Rules class; it defines lookup tables used to determine electrical characterization rules. This SKILL function does not create the table entries; to create table entries, use `techSetElectricalRuleTableEntry`. If a `tableCharacterizationRules` subclass does not exist, this function creates one.

Arguments

d_techFileId	The database identifier of the technology file.
t_ruleName	The name of the characterization rule.
l_indexDefinitions	A list defining the name, predefined index values, and user-defined match function for the table. The list has the following syntax: `( [t_index1Name] [l_def1Indices] [t_match1Func] [t_index2Name] [l_def2Indices] [t_match2Func] )` *t_index1Name* is the name of the first dimension of a two-dimensional table or the only dimension of a one-dimensional table. Valid Values: Any number or string *l_def1Indices* is a list of predefined indices for the first dimension, specifying the indices that can be used in the table. *t_match1Func* is the name of a user-defined match function. Valid Values: <, <=, =, >, >=, *t_userMatchFunction*, `nil`

where *userMatchFunction* is a user-defined function.
Default: ==

t_index2Name is the name of the second dimension of a two-dimensional table.
Valid Values: Any number or string

l_def2Indices is a list of predefined indices for the second dimension, specifying the indices that can be used in the table.

t_match2Func is the name of a user-defined match function.
Valid Values: <, <=, =, >, >=, *t_userMatchFunction*, nil
where *userMatchFunction* is a user-defined function.
Default: ==

*ltx_layer1*The first layer on which to apply the table rule.
Valid Values: the layer name, the layer number, a list containing the layer name and purpose

ltx_layer2 — The optional second layer on which to apply the table rule.
Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

t	The electrical rule table was created.
nil	The electrical rule table was not created.

Example

```
techCreateElectricalRuleTable( tfID "CAP" '("edgeCap" "parallelCap") "metal1" )
=> t
```

Creates the table electrical rule CAP with the indices edgeCap and parallelCap applied to layer metal1 in the technology file identified by tfID.

techGetElectricalRuleTable

```
techGetElectricalRuleTable(
     d_techFileId
     t_rule
     ltx_layer1
     [ltx_layer2]
     )
     => l_tables/nil
```

Description

Returns the table data for the specified table characterization rule defined in the `tableCharacterizationRules` subclass of the technology file and applied to the specified layer or layers. The `tableCharacterizationRules` subclass is located in the Electrical Rules class; it defines lookup tables used to determine electrical characterization rules.

Arguments

d_techFileId	The database identifier of the technology file.
t_rule	The name of the characterization rule.
ltx_layer1	The first layer on which to apply the table rule. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
ltx_layer2	The optional second layer on which to apply the table rule. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

l_tables	A list defining the table entries. The list has the following syntax: `( ( l_index1Definitions [l_index2Definitions] ) l_table )` *l_index1Definitions* is a list defining the name, predefined index values, and user-defined match function for the first table dimension. The format of the list is as follows:

`[nt_indexName] [(lg_indexValue...)] t_userMatchFunction]`

l_index2Definitions is a list defining the name, predefined index values, and user-defined match function for the first table dimension. The format of the list is as follows:

`[nt_indexName] [(lg_indexValue...)] [t_userMatchFunction]`

l_table is a list of the table entries.

`nil` The technology file does not exist or there are no electrical rule tables defined.

Example

```
techGetElectricalRuleTable( tfID "newRules" "metal1" "metal2" )
=> (
    ("one"(0.0 1.0 2.0) nil
     "two"(1.0 2.0 3.0) nil
    )
    (0.0 1.0) 0.0
    (1.0 2.0) 1.0
    (2.0 3.0) 2.0
)
```

Returns the table data for the `newRules` rule applied to `metal1` and `metal2` in the technology file identified by `tfID`. There are two indices, named `one` and `two`, and the table contains three index entries: `(0.0 1.0)` with associated value `0.0`, `(1.0 2.0)` with associated value `1.0`, and `(2.0 3.0)` with associated value `2.0`.

techGetElectricalRuleTables

```
techGetElectricalRuleTables(
     d_techFileId
     )
     => l_tables/nil
```

Description

Returns a list of the characterization rule tables defined in the `tableCharacterizationRules` subclass of the technology file. The `tableCharacterizationRules` subclass is located in the Electrical Rules class; it defines lookup tables used to determine electrical characterization rules.

Arguments

d_techFileId	The database identifier of the technology file.

Return Values

l_tables	A list of the table characterization rule names and the layers to which they apply. The list has the following syntax: `( ( t_rule ltx_layer1 [ltx_layer2] ) ... )` *t_rule* is the name of the spacing rule. Valid Values: any string *ltx_layer1* is the first layer on which the table rule is applied. Valid Values: the layer name, the layer number, a list containing the layer name and purpose *ltx_layer2* is the optional second layer on which the table rule is applied. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
`nil`	The technology file does not exist or there are no electrical rules tables defined.

Example

```
techGetElectricalRuleTables( tfID )
=> (("edgeCap" "metal1")
   ("cap1" "metal1" "metal2")
)
```

Returns the names of the table characterization rules and the layers to which each applies in the technology file identified by `tfID`. The first, `edgeCap`, is a single-layer rule applied to `metal1`; `cap1` is a two-layer rule applied to `metal1` and `metal2`.

techSetElectricalRuleTableEntry

```
techSetElectricalRuleTableEntry(
     d_techFileId
     t_rule
     l_index
     g_value
     ltx_layer1
     [ltx_layer2]
     )
     => t/nil
```

Description

Sets the value of the specified index in the specified characterization rule table defined in the `tableCharacterizationRules` subclass of the technology file. The `tableCharacterizationRules` subclass is located in the Electrical Rules class; it defines lookup tables used to determine electrical characterization rules.

Arguments

d_techFileId	The database identifier of the technology file.
t_rule	The name of the characterization rule.
l_index	The list of indices. The list has the following syntax: `( g_index1 [g_index2] )` *g_index1* is the first index in a two-dimensional table or the only index in a one-dimensional table. *g_index2* is the second index in a two-dimensional table.
g_value	The value to use when the application finds a match to the index or pair of indices during table lookup. Valid Values: Any number or string
ltx_layer1	The first layer on which the table rule is applied. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

ltx_layer2	The optional second layer on which the table rule is applied. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

t	The value was successfully set.
nil	The technology file does not exist, the table characterization rule is not defined, or the index is not defined.

Example

```
techSetElectricalRuleTableEntry( tfID "ACCURRENTDENSITY AVERAGE" 2 "5.0" "metal1")
=> t
```

Sets the value of the index 2 for the ACCURRENTDENSITY AVERAGE rule applied to metal1 in the technology file identified by tfID to 5.0.

techGetElectricalRuleTableEntry

```
techGetElectricalRuleTableEntry(
     d_techFileId
     t_rule
     l_index
     ltx_layer1
     [ltx_layer2]
     )
     => g_value/nil
```

Description

Returns the value of the specified index in the specified characterization rule table defined in the `tableCharacterizationRules` subclass of the technology file. The `tableCharacterizationRules` subclass is located in the Electrical Rules class; it defines lookup tables used to determine electrical characterization rules.

Arguments

d_techFileId	The database identifier of the technology file.
t_rule	The name of the characterization rule.
l_index	The list of indices. The list has the following syntax: `( g_index1 [g_index2] )` *g_index1* is the first index in a two-dimensional table or the only index in a one-dimensional table. *g_index2* is the second index in a two-dimensional table.
ltx_layer1	The first layer on which the table rule is applied. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
ltx_layer2	The optional second layer on which the table rule is applied. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

g_value	The value of the specified index.
nil	The technology file does not exist or the table characterization rule is not defined.

Example

```
techGetElectricalRuleTableEntry( tfID "AREA CAP" 2 "metal1" )
=> 0.0
```

Returns the value (0.0) of the index 2 for the table rule AREA CAP applied to the layer metal1 in the technology file identified by tfID.

5

Physical Design Application SKILL Functions

This chapter describes the Cadence® SKILL language functions you can use to create and update the rules classes that control how the physical design applications work. The SKILL functions for the physical design applications include the following:

- Virtuoso Layout Editor SKILL Functions
- Virtuoso XL Layout Editor SKILL Functions
- Virtuoso Compactor SKILL Functions

Virtuoso Layout Editor SKILL Functions

The functions described in this section operate on the subclass of the Layout Editor Rules class in the technology file. The Layout Editor Rules class includes a subclass that lets you specify the layers that are displayed in the Layer Selection Window (LSW). In the subclass, you list the layers in the order you want them to appear in the LSW.

If you do not define the `leLswLayers` subclass in the technology file, the Virtuoso® Layout Editor uses the `techLayerPurposePriorities` subclass of the Layer Definitions class to determine the layers and order of the LSW.

Note: The `leLswLayers` subclass specifies the layers that appears in the LSW. The LSW provides commands that let you modify visibility and selectability of the layers as you are working.

techSetLeLswLayers

```
techSetLeLswLayers(
     d_techFileID
     l_lswLayers
     )
     => t/nil
```

Description

Creates an `leLswLayers` subclass in the specified technology file. The `leLswLayers` subclass is located in the Layout Editor Rules class; it lists the layer-purpose pairs that are displayed in your designs in the order in which they appear in the LSW. If an `leLswLayers` subclass already exists, this function deletes it and replaces it with the specified data.

For more information about the `leLswLayers` subclass of the technology file, refer to "leLswLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
l_lswLayers	An ordered list of layers to display in the LSW. The list has the following syntax: `list( l_layer ... )` *l_layer* is a list containing the layer name and purpose.

Return Values

`t`	The `leLSWLayers` subclass in the specified technology file was created or replaced.
`nil`	The technology file does not exist.

Example

```
techSetLeLswLayers( techFileID
list( ( "metal1" "drawing" )
( "metal2" "drawing" )
( "metal3" "drawing" )
( "poly1" "drawing" )
( "pWell" "drawing" )
( "implant" "drawing" )
( "diff" "drawing" )
( "align" "drawing" ))
)
=> t
```

Creates an `leLswLayers` subclass with these layers in the technology file identified by `techFileID`.

techSetLeLswLayer

```
techSetLeLswLayer(
     d_techFileID
     l_layer
     )
     => t/nil
```

Description

Appends the specified layer to the end of the `leLswLayers` subclass in the specified technology file. The `leLswLayers` subclass is located in the Layout Editor Rules class; it lists the layer-purpose pairs that are displayed in your designs in the order in which they appear in the LSW. If the `leLswLayers` subclass does not exist, this function creates it with the specified data.

For more information about the `leLswLayers` subclass of the technology file, refer to "leLswLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`l_layer`	The layer-purpose pair to append to the `leLswLayers` subclass. Valid Values: a list containing the layer name and purpose

Return Values

`t`	The layer was added to the `leLswLayers` subclass in the specified technology file.
`nil`	The technology file does not exist.

Example

```
techSetLeLswLayer(techFileID ("nwell" "drawing"))
=> t
```

Appends the `nwell drawing` layer to the `leLswLayers` subclass in the Layout Editor Rules class of the technology file identified by `techFileID`.

techGetLeLswLayers

```
techGetLeLswLayers(
     d_techFileID
     )
     => l_lswLayers/nil
```

Description

Returns the layer-purpose pairs listed in the `leLswLayers` subclass of the Layout Editor Rules class of the specified technology file. The `leLswLayers` subclass lists the layer-purpose pairs that are displayed in your designs in the order in which they appear in the LSW.

For more information about the `leLswLayers` subclass of the technology file, refer to "leLswLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_lswLayers	A list of the layers specified by the `leLswLayers` subclass. The list has the following syntax: `( (t_layerName t_layerPurpose) ... )`
`nil`	The technology file or the `leLswLayers` subclass does not exist.

Example

```
techGetLeLswLayers( techFileID )
=>(( "metal1" "drawing" )
( "metal2" "drawing" )
( "metal3" "drawing" )
( "poly1" "drawing" )
( "pWell" "drawing" )
( "implant" "drawing" )
( "diff" "drawing" )
( "align" "drawing" ))
```

Returns the layers defined in the `leLswLayers` subclass of the Layout Editor Rules class of the technology file identified by `techFileID`.

techIsLeLswLayer

```
techIsLeLswLayer(
     d_techFileID
     l_layer
     )
     => t/nil
```

Description

Indicates whether the specified layer-purpose pair is listed in the `leLswLayers` subclass of the technology file. The `leLswLayers` subclass is located in the Layout Editor Rules class; it lists the layer-purpose pairs that are displayed in your designs in the order in which they appear in the LSW.

For more information about the `leLswLayers` subclass of the technology file, refer to "leLswLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
l_layer	The layer-purpose pair to check. Valid Values: a list containing the layer name and purpose

Return Values

`t`	The specified layer is listed in the `leLswLayers` subclass in the specified technology file.
`nil`	The technology file does not exist or the layer-purpose pair is not listed in the `leLswLayers` subclass.

Example

```
techIsLeLswLayer( techFileID "metal1" )
=> t
```

The `metal1` layer is listed in the `leLswLayers` subclass in the Layout Editor Rules class of the technology file identified by `techFileID`.

Virtuoso XL Layout Editor SKILL Functions

The functions described in this section operate on the subclasses of the Virtuoso XL Rules class in the technology file. The Virtuoso XL Rules class includes subclasses that let you specify which layers you want the Virtuoso XL Layout Editor (Virtuoso XL) to monitor and which layers cannot overlap in a design.

You must define these subclasses in the Virtuoso XL Rules class before you can use the layout accelerator to develop the connectivity of a design.

techSetLxExtractLayers

```
techSetLxExtractLayers(
     d_techFileID
     l_extractLayers
     )
     => t/nil
```

Description

Creates an `lxExtractLayers` subclass in the specified technology file. The `lxExtractLayers` subclass is located in the Virtuoso XL Rules class; it lists the layers the online extractor monitors. If an `lxExtractLayers` subclass already exists, this function deletes it and replaces it with the specified data.

For more information about the `lxExtractLayers` subclass of the technology file, refer to "lxExtractLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
l_extractLayers	A list of layers monitored by the online extractor. If you specify only a layer name or number, all layer-purpose pairs of that layer are monitored. The list has the following syntax: `list( ltx_layer ... )` *ltx_layer* is a list containing the layer name and purpose, or the layer name, or the layer number.

Return Values

`t`	The `lxExtractLayers` subclass in the specified technology file was created or replaced.
`nil`	The technology file does not exist.

Example

```
techSetLxExtractLayers( techFileID list( "metal1" "poly" "metal2" )
=> t
```

Creates an `lxExtractLayers` subclass with the specified layers in the technology file identified by `techFileID`.

techSetLxExtractLayer

```
techSetLxExtractLayer(
     d_techFileID
     ltx_extractLayer
     )
     => t/nil
```

Description

Appends the specified layer to the end of the `lxExtractLayers` subclass in the specified technology file. The `lxExtractLayers` subclass is located in the Virtuoso XL Rules class; it lists the layers the online extractor monitors. If the `lxExtractLayers` subclass does not exist, this function creates it with the specified data.

For more information about the `lxExtractLayers` subclass of the technology file, refer to "lxExtractLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`ltx_extractLayer`	A list containing the layer name and purpose, or the layer name, or the layer number. If you specify only a layer name or number, all layer-purpose pairs of that layer are monitored.

Return Values

`t`	The layer was added to the `lxExtractLayers` subclass in the specified technology file.
`nil`	The technology file does not exist.

Example

```
techSetLxExtractLayer( techFileID "metal1" )
=> t
```

Appends the `metal1` layer to the `lxExtractLayers` subclass in the Virtuoso XL Rules class of the technology file identified by `techFileID`.

techGetLxExtractLayers

```
techGetLxExtractLayers(
    d_techFileID
    )
    => l_extractLayers/nil
```

Description

Returns the layers listed in the `lxExtractLayers` subclass of the Virtuoso XL Rules class of the specified technology file. The `lxExtractLayers` subclass lists the layers the online extractor monitors.

For more information about the `lxExtractLayers` subclass of the technology file, refer to "lxExtractLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_extractLayers	A list of the layers specified by the `lxExtractLayers` subclass. The list has the following syntax: `list ( lt_layer ... )` *lt_layer* is the layer data as it is stored in the `lxExtractLayers` subclass of the specified technology file. The layer can be stored as the layer name or a list of the layer and purpose.
`nil`	The technology file or the `lxExtractLayers` subclass does not exist.

Example

```
techGetLxExtractLayers( techFileID )
=> ( pWell nDiff pDiff poly1 cont metal1 via1 metal2 via2 metal3 )
```

Returns the layers specified in the `lxExtractLayers` subclass of the Virtuoso XL Rules class of the technology file identified by `techFileID`.

techIsLxExtractLayer

```
techIsLxExtractLayer(
     d_techFileID
     ltx_layer
     )
     => t/nil
```

Description

Indicates whether the specified layer is listed in the technology file as an extract layer. The `lxExtractLayers` subclass is located in the Virtuoso XL Rules class; it lists the layers the online extractor monitors.

For more information about the `lxExtractLayers` subclass of the technology file, refer to "lxExtractLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`ltx_layer`	The layer to check. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

`t`	The specified layer is listed in the `lxExtractLayers` subclass in the specified technology file.
`nil`	The technology file does not exist, or the layer is not listed in the `lxExtractLayers` subclass.

Example

```
techIsLxExtractLayer( techFileID "metal1" )
=> t
```

The `metal1` layer is listed in the `lxExtractLayers` subclass in the Virtuoso XL Rules class of the technology file identified by `techFileID`.

techSetLxNoOverlapLayers

```
techSetLxNoOverlapLayers(
     d_techFileID
     l_noOverlapLayers
     )
     => t/nil
```

Description

Creates an `lxNoOverlapLayers` subclass in the specified technology file. The `lxNoOverlapLayers` subclass is located in the Virtuoso XL Rules class; it lists sets of layers that cannot overlap in a Virtuoso XL design without producing an error. If an `lxNoOverlapLayers` subclass already exists, this function deletes it and replaces it with the specified data.

For more information about the `lxNoOverlapLayers` subclass of the technology file, refer to "lxNoOverlapLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
l_noOverlapLayers	A list of lists of layers that cannot overlap. Layer lists must contain at least two layers. The list has the following syntax: `list( list(`*ltx_layer* `...) ... )` *ltx_layer* is a list containing the layer name and purpose, or the layer name, or the layer number.

Return Values

`t`	The `lxNoOverlapLayers` subclass in the specified technology file was created or replaced.
`nil`	The technology file does not exist.

Example

```
techSetLxNoOverlapLayers( techFileID
list(
list( poly1 nDiff )
list( poly1 pDiff )
list( via1  via2 )
)
=> t
```

Creates an `lxNoOverlapLayers` subclass with the specified layers in the technology file identified by `techFileID`.

techSetLxNoOverlapLayer

```
techSetLxNoOverlapLayer(
     d_techFileID
     l_noOverlapLayers
     )
     => t/nil
```

Description

Appends the specified layer to the end of the `lxNoOverlapLayers` subclass in the specified technology file. The `lxNoOverlapLayers` subclass is located in the Virtuoso XL Rules class; it lists sets of layers that cannot overlap in a Virtuoso XL design without producing an error. If the `lxNoOverlapLayers` subclass does not exist, this function creates it with the specified data.

For more information about the `lxNoOverlapLayers` subclass of the technology file, refer to "lxNoOverlapLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
l_noOverlapLayers	A list of at least two layers that cannot overlap. The list has the following syntax: `list( ltx_layer ... )` *ltx_layer* is the layer name, layer number, or a list containing the layer name and purpose.

Return Values

`t`	The list of layers was added to the `lxNoOverlapLayers` subclass in the specified technology file.
`nil`	The technology file does not exist.

Example

```
techSetLxNoOverlapLayer( techFileID (via1 via2) )
=> t
```

Appends the list of `via1` and `via2` to the `lxNoOverlapLayers` subclass in the Virtuoso XL Rules class of the technology file identified by `techFileID`.

techGetLxNoOverlapLayers

```
techGetLxNoOverlapLayers(
     d_techFileID
     )
     => l_noOverlapLayers/nil
```

Description

Returns the layers listed in the `lxNoOverlapLayers` subclass of the specified technology file. The `lxNoOverlapLayers` subclass is located in the Virtuoso XL Rules class; it lists sets of layers that cannot overlap in a Virtuoso XL design without producing an error.

For more information about the `lxNoOverlapLayers` subclass of the technology file, refer to "lxNoOverlapLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_noOverlapLayers	A list of the layers specified by the `lxNoOverlapLayers` subclass. The list has the following syntax: `( (`*lt_layer* `... ) ... )` *lt_layer* is the layer data as it is stored in the `lxNoOverlapLayers` subclass of the specified technology file. The layer can be stored as the layer name or a list of the layer name and purpose.
`nil`	The technology file or the `lxNoOverlapLayers` subclass does not exist.

Example

```
techGetLxNoOverlapLayers( techFileID )
=> (( poly1 nDiff ) ( poly1 pDiff ) ( via1  via2 ))
```

Returns the layer lists from the `lxNoOverlapLayers` subclass of the Virtuoso XL Rules class of the technology file identified by `techFileID`.

techIsLxNoOverlapLayer

```
techIsLxNoOverlapLayer(
     d_techFileID
     l_noOverlapLayers
     )
     => t/nil
```

Description

Indicates whether the specified set of layers is listed in the `lxNoOverlapLayers` subclass of the technology file. The `lxNoOverlapLayers` subclass is located in the Virtuoso XL Rules class; it lists sets of layers that cannot overlap in a Virtuoso XL design without producing an error.

For more information about the `lxNoOverlapLayers` subclass of the technology file, refer to "lxNoOverlapLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
l_noOverlapLayers	The list of layers to check. The list has the following syntax: `list( ltx_layer ... )` *ltx_layer* is the layer name, layer number, or a list containing the layer name and purpose.

Return Values

`t`	The specified layer is listed in the `lxNoOverlapLayers` subclass in the specified technology file.
`nil`	The technology file does not exist or the layer is not listed in the `lxNoOverlapLayers` subclass.

Example

```
techIsLxNoOverlapLayer( techFileID
'("via1" "via2") )
=> t
```

The set of `via1` and `via2` is listed in the `lxNoOverlapLayers` subclass in the Virtuoso XL Rules class of the technology file identified by `techFileID`.

techSetMPPTemplate

```
techSetMPPTemplate(
               (    d_techFileId
                    t_mppTemplateName
                    l_template
               ) ;end of template
               ) ;end of techSetMPPTemplate

               ; 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]
            [ for subrectangles]
        ) ; 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 single template in your technology library in virtual memory that specifies a relative object design (ROD) multipart path (MPP). 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. The purpose of an MPP template is to let you create MPPs in layout cellviews using predefined values from your technology library. You can define any number of MPP templates in your

technology library; each template must be identified by a unique template name (*t_mppTemplateName*).

Note: Adding and deleting templates affects 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 your binary technology library on disk before you exit the software.

Arguments

d_techFileId	Database identifier for the technology file in which you want to define an MPP template.
t_mppTemplateName	Character string enclosed in double quotation marks specifying the name of the MPP template. The name must be unique within the MPP templates in your technology library. 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, `nil`. Specifying `nil` deletes the template specified by *t_mppTemplateName* from your technology library.

The syntax for `techSetMPPTemplate` is based on the syntax of the `rodCreatePath` function. Most arguments are the same and have the same argument definitions, valid values, and default values; the exceptions are described below. For a detailed description of the arguments, see the "Arguments" section of the `rodCreatePath` function in the *Virtuoso Relative Object Design User Guide*.

The syntax for `techSetMPPTemplate` and `rodCreatePath` differ as follows:

- Arguments for `techSetMPPTemplate` are positional; you must specify them in the sequence shown in the documentation for the `techSetMPPTemplate` syntax. Arguments for the `rodCreatePath` function are key-word value pairs; you can specify them in any sequence.
- `techSetMPPTemplate` has two additional arguments: *d_techFileId* and *t_mppTemplateName*.
- For `techSetMPPTemplate`, all arguments other than *d_techFileId* and *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 `techSetMPPTemplate` 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 `techSetMPPTemplate` arguments, you must enter a character string (the data type is *`t_`* for text).
- `techSetMPPTemplate` does not contain the following `rodCreatePath` arguments because their values 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 system uses the value of *Net Name* for terminal name.

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

- `techSetMPPTemplate` 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_fromObjtxf_size`
  `l_startHandle` *`l_endHandle`*

For detailed information about the `rodCreatePath` function, its arguments, valid values, and default values, see `rodCreatePath` in the *Virtuoso Relative Object Design User Guide*.

In the `lxMPPTemplates` subclass section of this document, the following topics contain information that also applies to `techSetMPPTemplate`:

- Specifying Positional Arguments
- Specifying Arguments as nil
- Specifying Template Arguments as SKILL Expressions
- Example of an MPP Template Definition

techGetMPPTemplateNames

```
techGetMPPTemplateNames(
     d_techFileId
     )
     => l_mppTemplateNames/nil
```

Description

Returns the names of all multipart path (MPP) templates defined in the current technology library.

Arguments

d_techFileId	The database identifier of the technology file.

Return Values

l_mppTemplateNames	A list of the names of the MPP templates defined in the technology library.
nil	The technology file does not exist or the technology library does not contain MPP template definitions.

Example

```
techGetMPPTemplateNames( techFileID )
=> ("guardring" "template2")
```

Returns the names of the MPP templates defined in the technology library containing the technology file identified by techFileID.

techGetMPPTemplateByName

```
techGetMPPTemplateByName(
     d_techFileId
     t_mppTemplateName
     )
     => l_template/nil
```

Description

Returns the definition of the specified multipart path (MPP) template in the technology library containing the technology file identified by *d_techFileId*.

Arguments

d_techFileId	The database identifier of the technology file.
t_mppTemplateName	The name of the MPP template to retrieve.

Return Values

l_template	A list defining the MPP template. The elements of this list match the *l_template* arguments for techSetMPPTemplate.
nil	The technology file does not exist or the technology library does not contain the named MPP template definition.

Example

```
techGetMPPTemplateByName( (tf "guardring")

=> ((("ndiff" "drawing") 1.8 nil) nil
    ((("metal1" "drawing") 0.2 t -0.2 0.2
         ("inputOutput" t
             ("left" "right" "top" "bottom") t 1.0
             ("text" "drawing") "centerCenter" "stick" t "R0"
             "lowerLeft"
             (0.0 0.0)
         )
    )
   )
   ((("cont" "drawing") nil nil t nil
        nil nil -0.6
    )
   )
)
```

Returns the definition of the MPP template named `guardring` in the technology library containing the technology file identified by `tf`.

Virtuoso Compactor SKILL Functions

The functions described in this section operate on the subclasses of the Compactor Rules class in the technology file. The Compactor Rules class includes subclasses that let you specify the layers and define the wires and rules the Virtuoso Compactor uses to compact layout designs. You must define these subclasses before you can use the compactor to generate layout designs.

techSetCompactorLayers

```
techSetCompactorLayers(
     d_techFileID
     l_compactorLayers
     )
     => t/nil
```

Description

Creates a `compactorLayers` subclass in the specified technology file. The `compactorLayers` subclass is located in the Compactor Rules class; it lists the layers and how the Virtuoso Compactor processes them when it compacts a design. If a `compactorLayers` subclass already exists, this function deletes it and replaces it with the specified data.

For more information about the `compactorLayers` subclass of the technology file, refer to "compactorLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
l_compactorLayers	A list of the layers and the compactor usage keywords you want to define. The list has the following syntax: `list( list( ltx_layer t_usage ) ... )` *ltx_layer* specifies the layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose *t_usage* is the keyword that indicates how you want the compactor to use the layer. Valid Values: `diffusion`, `conduction`, `via`, `well`, `implant`, `hardFence`, `softFence`, `cellBoundary`

Return Values

`t`	The `compactorLayers` subclass in the specified technology file was created or replaced.

`nil` The technology file does not exist.

Example

```
techSetCompactorLayers( techFileID list(
list( "diff" "diffusion" )
list( "poly1" "conduction" )
list( "metal" "conduction" )
list( "metal1" "conduction" )
list( "metal2" "conduction" )
list( "via1" "via" )
list( "via2" "via" )
list( "pWell" "well" )
list( "implant" "implant" )
list( "hardFence" "hardFence" )
list( "softFence" "softFence")
list( "bndry" "cellBoundary" )
) )
=> t
```

Creates a `compactorLayers` subclass with the specified layer and usage pairs in the technology file identified by `techFileID`.

techSetCompactorLayer

```
techSetCompactorLayer(
     d_techFileID
     ltx_compactorLayer
     t_usage
     )
     => t/nil
```

Description

Updates the usage of the specified layer in the `compactorLayers` subclass of the specified technology file. The `compactorLayers` subclass is located in the Compactor Rules class; it lists the layers and how the compactor processes them when it compacts a design. If the layer is not listed in the `compactorLayers` subclass, this function appends the layer name and usage. If the `compactorLayers` subclass does not exist, this function creates it with the specified data.

For more information about the `compactorLayers` subclass of the technology file, refer to "compactorLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`ltx_compactorLayer`	The layer for which to update the usage. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
`t_usage`	The keyword that indicates how you want the compactor to use the layer. Valid Values: `diffusion`, `conduction`, `via`, `well`, `implant`, `hardFence`, `softFence`, `cellBoundary`

Return Values

`t`	The data was added to the `compactorLayers` subclass in the specified technology file.
`nil`	The technology file does not exist.

Example

```
techSetCompactorLayer( techFileID "metal3" "conduction" )
=> t
```

Appends or updates the specified layer in the `compactorLayers` subclass in the Compactor Rules class of the technology file identified by `techFileID`.

techGetCompactorLayers

```
techGetCompactorLayers(
     d_techFileID
     )
     => ltx_compactorLayers/nil
```

Description

Returns a list of the layer and usage data defined in the `compactorLayers` subclass the specified technology file. The `compactorLayers` subclass is located in the Compactor Rules class; it lists the layers and how the compactor processes them when it compacts a design.

For more information about the `compactorLayers` subclass of the technology file, refer to "compactorLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

ltx_compactorLayers	A list of the layer and usage data specified by the `compactorLayers` subclass. The list has the following syntax: `list ( list( lt_layer t_usage ) ... )` *lt_layer* is the layer data as it is stored in the `compactorLayers` subclass of the specified technology file. The layer could be stored as the layer name or a list of the layer and purpose. *t_usage* is the usage assigned to the layer.
`nil`	The technology file or the `compactorLayers` subclass does not exist.

Example

```
techGetCompactorLayers( techFileID )
=> (
( "diff" "diffusion" )
( "poly1" "conduction" )
( "metal" "conduction" )
( "metal1" "conduction" )
( "metal2" "conduction" )
( "via1" "via" )
( "via2" "via" )
( "pWell" "well" )
( "implant" "implant" )
( "hardFence" "hardFence" )
( "softFence" "softFence" )
( "bndry" "cellBoundary" )
)
```

Returns the compactor layer data defined by the `compactorLayers` subclass in the Compactor Rules class of the technology file identified by `techFileID`.

techGetCompactorUsage

```
techGetCompactorUsage(
     d_techFileID
     ltx_layer
     )
     => t_usage/nil
```

Description

Returns the usage data defined for the specified layer in the `compactorLayers` subclass of the specified technology file. The `compactorLayers` subclass is located in the Compactor Rules class; it lists the layers and how the compactor processes them when it compacts a design.

For more information about the `compactorLayers` subclass of the technology file, refer to "compactorLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
ltx_layer	The layer for which you want to get the usage. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

t_usage	The usage assigned to the specified layer.
`nil`	The technology file does not exist, the `compactorLayers` subclass does not exist, or the layer is not defined in the `compactorLayers` subclass.

Example

```
techGetCompactorUsage( techFileID "poly1" )
=> conduction
```

Returns the usage of the `poly1` layer defined by the `compactorLayers` subclass in the Compactor Rules class of the technology file identified by `techFileID`.

techIsCompactorLayer

```
techIsCompactorLayer(
     d_techFileID
     ltx_layer
     )
     => t/nil
```

Description

Indicates whether the specified layer is listed in the `compactorLayers` subclass of the technology file.The `compactorLayers` subclass is located in the Compactor Rules class; it lists the layers and how the compactor processes them when it compacts a design.

For more information about the `compactorLayers` subclass of the technology file, refer to "compactorLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`ltx_layer`	The layer to check. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

`t`	The specified layer is listed in the `compactorLayers` subclass in the specified technology file.
`nil`	The technology file does not exist, the `compactorLayers` subclass does not exist, or the layer is not listed in the `compactorLayers` subclass.

Example

```
techIsCompactorLayer( techFileID "poly1" )
=> t
```

The `poly1` layer is listed in the `compactorLayers` subclass in the Compactor Rules class of the technology file identified by `techFileID`.

techSetSymWire

```
techSetSymWire(
     d_techFileID
     t_wireName
     l_layer
     [l_implant]/nil
     [l_width]/nil
     [l_region]/nil
     n_wlm_weight
     )
     => t/nil
```

Description

Updates the parameters of the specified symbolic wire in the `symWires` subclass of the specified technology file. The `symWires` subclass is located in the Compactor Rules class; it lists the wires and how the compactor processes them when it compacts a design. If the wire is not listed in the `symWires` subclass, this function adds the wire. If the `symWires` subclass does not exist, this function creates it with the specified data.

For more information about the `symWires` subclass of the technology file, refer to "symWires" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_wireName	The wire name to update or create.
l_layer	The layer for the wire. Valid Values: a list containing the layer name and purpose
l_implant	A list containing the implant layer and implant spacing, or `nil` if you do not want to define implant layers but do want to define width or legal region data. The syntax of these arguments is as follows: *ltx_layer* is the layer name, or the layer number, or a list containing the layer name and layer purpose. Layer purpose defaults to `drawing` if not specified.

	g_spacing is the distance, in user units, by which the implant layer must enclose the wire layer.
l_width	A list containing the default, minimum, and maximum widths of the wire, or `nil` if you do not want to define width data but do want to define legal region data. At least one of the arguments must be specified as an integer. The syntax of these arguments is as follows: *g_default* is the default width of the wire, in user units. Valid Values: an integer, `nil` *g_minWidth* is the minimum width allowed for the wire, in user units. Valid Values: an integer, `nil` *g_maxWidth* is the maximum width allowed for the wire, in user units. Valid Values: an integer, `nil`
l_region	A list containing the legal region and region layer for the wire. The syntax of these arguments is as follows: *t_regionName* is the name of the legal region. Valid Values: `inside`, `outside` *ltx_layer* is the layer-purpose pair used to draw the legal region. The layer purpose defaults to `drawing` if you supply a layer name or number. Valid Values: the layer name, the layer number, a list containing the layer name and layer purpose
n_wlm_weight	Weighting indicating the relative priority to the compactor for minimizing the length of the wire. Valid Values: any positive number

Return Values

`t`	The data was updated or added to the `symWires` subclass in the specified technology file.
`nil`	The technology file does not exist.

Example

```
techSetSymWire( techFileID "MET1"  list("metal1" "drawing") nil list(1.0 0.8 nil)
nil )
=> t
```

Updates the layer and width data for the specified wire in the `symWires` subclass in the Compactor Rules class of the technology file identified by `techFileID`.

techGetSymWires

```
techGetSymWires(
     d_techFileID
     )
     => l_symWires/nil
```

Description

Returns the list of the symbolic wires defined in the `symWires` subclass of the specified technology file. The `symWires` subclass is located in the Compactor Rules class; it lists the wires and how the compactor processes them when it compacts a design.

For more information about the `symWires` subclass of the technology file, refer to "symWires" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_symWires	A list of the symbolic wires defined in the specified technology file. The list has the following syntax: `( t_wireName ... )` *t_wireName* is the wire name to update or create.
`nil`	The technology file or the `symWires` subclass does not exist.

Example

```
techGetSymWires( techFileID )
=> ( "diff" "poly1" "metal" "metal1" "metal2" )
```

Returns the names of the symbolic wires defined by the `symWires` subclass in the Compactor Rules class of the technology file identified by `techFileID`.

techGetSymWireParams

```
techGetSymWireParams(
     d_techFileID
     t_wireName
     )
     => l_wireParams/nil
```

Description

Returns the parameters of the specified symbolic wire defined in the `symWires` subclass of the specified technology file. The `symWires` subclass is located in the Compactor Rules class; it lists the wires and how the compactor processes them when it compacts a design.

For more information about the `symWires` subclass of the technology file, refer to "symWires" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_wireName	The name of a wire defined in the specified technology file.

Return Values

l_wireParams	A list of the parameters for the specified symbolic wire in the `symWires` subclass. The list has the following syntax:
	`( t_wireName ( t_layerName t_layerPurpose )`
	`( lt_implantLayer g_implantSpacing )|nil`
	`( g_defaultWidth g_minWidth g_maxWidth )|`
	`nil [( t_regionName lt_regionLayer g_wlm_weight )] )`
	
	t_wireName is the name of the wire.
	
	t_layerName is the name of the layer for the wire.
	
	t_layerPurpose is the purpose of the layer for the wire.
	
	lt_implantLayer specifies the layer data stored for the implant layer. This could be the layer name or a list containing the layer name and layer purpose.

g_implantSpacing is the distance, in user units, by which the implant layer must enclose the wire layer.

g_defaultWidth is the default width of the wire, in user units.

g_minWidth is the minimum width allowed for the wire, in user units.

g_maxWidth is the maximum width allowed for the wire, in user units.

t_regionName is the name of the legal region.

lt_regionLayer specifies the layer data stored for the legal region. This could be the layer name or a list containing the layer name and layer purpose.

g_wlm_weight is a weighting indicating the relative priority to the compactor for minimizing the length of the wire.
Valid Values: any positive number

`nil` The technology file does not exist, the `symWires` subclass does not exist, or the symbolic wire is not defined in the `symWires` subclass.

Example

```
techGetSymWireParams( techFileID "pdiff" )
=> ( "pdiff" ("diff" "drawing") ( ("implant" "drawing").2 )
(.6 .6 .6) ("inside" "nwell") )
```

Returns the parameters of the `pdiff` symbolic wire defined by the `symWires` subclass in the Compactor Rules class of the technology file identified by `techFileID`.

techSetSymRules

```
techSetSymRules(
     d_techFileID
     l_symRules
     )
     => t/nil
```

Description

Creates a `symRules` subclass in the specified technology file. The `symRules` subclass is located in the Compactor Rules class; it lists the rules that the compactor uses when it compacts a design. Symbolic rules are different from physical rules in that you can restrict design rules to objects on the same net or on different nets, or to layers on specific devices or parameterized cells. If a `symRules` subclass already exists, this function deletes it and replaces it with the specified data.

For more information about the `symRules` subclass of the technology file, refer to "symWires" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
l_symRules	A list of the symbolic rules that you want to define. The list has the following syntax: `list( list( "drc" ltx_layerDevice [ltx_layerDevice] ( t_ruleType < n_rule ) [t_modifier] ) ... )`

For detailed information about setting symbolic rules, refer to "symWires" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Return Values

`t`	The `symRules` subclass was created or replaced in the specified technology file.
`nil`	The technology file does not exist.

Example

```
techSetSymRules( techFileID
'((drc
     ("diff" "drawing" "NTR")
     ("diff" "drawing" PTAP")
     (sep < 0.3)
)))
=> t
```

Sets a design rule that specifies that the `diff` layer on `NTR` devices must be less than `0.3` user units from the `diff drawing` layer on `PTAP` devices.

techGetSymRules

```
techGetSymRules(
     d_techFileID
     )
     => l_symRules/nil
```

Description

Returns a list of the symbolic rules defined in the `symRules` subclass of the specified technology file. The `symRules` subclass is located in the Compactor Rules class; it lists the rules the compactor uses when it compacts a design.

For more information about the `symRules` subclass of the technology file, refer to "symWires" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_symRules	A list of the `symRules` defined in the specified technology file.
`nil`	The technology file or the `symRules` subclass does not exist.

Example

```
techGetSymRules( techFileID )
=> ( (drc(("pimplant" "drawing" "PTAP") ("diff" "drawing" "NTR") (sep < 0.900000)))
(drc(("pimplant" "drawing" "PTAP") ("diff" "drawing" "M1_N") (sep < 0.900000)))
(drc(("diff" "drawing" "NTAP") ("pimplant" "drawing" "PTR") (sep < 0.900000)))
(drc(("diff" "drawing" "NTAP") ("pimplant" "drawing" "M1_P") (sep < 0.900000)))
(drc(( "cont" "drawing" "M1_POLY1") ("diff" "drawing" "NTR") sep <.6 ))
(drc(( "cont" "drawing" "M1_POLY1") ("diff" "drawing" "PTR") sep <.6 ))
)
```

Returns the rules defined in the `symRules` subclass in the Compactor Rules class of the technology file identified by `techFileID`.

6

Place and Route Application SKILL Functions

This chapter describes the SKILL functions you can use to create and update the Place and Route Rules class. This class contains rules that control how the place-and-route applications work.

The following Cadence® SKILL language functions apply to place and route applications such as Preview Silicon Ensemble™ and Preview Gate Ensemble®:

techSetPrRoutingLayers

techSetPrRoutingLayer

techGetPrRoutingLayers

techGetPrRoutingDirection

techIsPrRoutingLayer

techSetPrViaTypes

techSetPrViaType

techGetPrViaTypes

techGetPrViaType

techIsPrViaDevice

The following SKILL functions apply to Preview Silicon Ensemble and Preview Gate Ensemble:

techSetPrStackVias

techSetPrStackVia

techGetPrStackVias

techIsPrStackVia
techSetPrMaxStackVia
techSetPrMaxStackVias
techSetPrMastersliceLayers
techSetPrMastersliceLayer
techGetPrMastersliceLayers
techIsPrMastersliceLayer
techSetPrViaRule
techGetPrViaRules
techGetPrViaParams
techSetPrGenViaRule
techGetPrGenViaRules
techGetPrGenViaParams
techSetPrNonDefaultRule
techGetPrNonDefaultRules
techGetPrNonDefaultParams

The following SKILL functions apply to Preview Silicon Ensemble only:

techSetPrRoutingPitch
techGetPrRoutingPitch
techSetPrRoutingOffset
techGetPrRoutingOffset
techSetPrOverlapLayer
techSetPrOverlapLayers
techGetPrOverlapLayers

techSetPrRoutingLayers

```
techSetPrRoutingLayers(
     d_techFileID
     l_routingLayers
     )
     => t/nil
```

Description

Creates a `prRoutingLayers` subclass in the specified technology file. The `prRoutingLayers` subclass is located in the Place and Route Rules class; it lists the routing layers in the order of placement. Layers closest to the substrate are listed first. If a `prRoutingLayers` subclass already exists, this function deletes it and replaces it with the specified data.

For more information about the `prRoutingLayers` subclass of the technology file, refer to "prRoutingLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`l_routingLayers`	A list of the layers and the optional direction keywords you want to define. The list has the following syntax: `list( list( ltx_layer t_direction ) ... )` *`ltx_layer`* is the layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose *`t_direction`* is the keyword that indicates the primary direction to route the layer. Valid Values: `halfRoute`, `horizontal`, `vertical`

Return Values

`t`	The `prRoutingLayers` subclass in the specified technology file was created or replaced.
`nil`	The technology file does not exist.

Example

```
techSetPrRoutingLayers( techFileID list(
list( "poly1" "halfRoute" )
list( "metal1" "horizontal" )
list( "metal2" "vertical" )
list( "metal" "horizontal" )
)
)
=> t
```

Creates a `prRoutingLayers` subclass with the specified layer-direction pairs in the technology file identified by `techFileID`.

techSetPrRoutingLayer

```
techSetPrRoutingLayer(
     d_techFileID
     ltx_routingLayer
     t_direction
     )
     => t/nil
```

Description

Updates the direction of the specified layer in the `prRoutingLayers` subclass in the specified technology file. The `prRoutingLayers` subclass is located in the Place and Route Rules class; it lists the layers in the order of placement. Layers closest to the substrate are listed first. If the layer is not listed in the `prRoutingLayers` subclass, this function appends the layer name and direction. If the `prRoutingLayers` subclass does not exist, this function creates it with the specified data.

For more information about the `prRoutingLayers` subclass of the technology file, refer to "prRoutingLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`ltx_routinglayer`	The layer for which to update the direction. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
`t_direction`	The keyword that indicates how you want the place-and-route tools to use the layer. Valid Values: `halfRoute`, `horizontal`, `vertical`

Return Values

`t`	The data was added to the `prRoutingLayers` subclass in the specified technology file.
`nil`	The technology file does not exist.

Example

```
techSetPrRoutingLayer( techFileID "poly" "halfRoute" )
=> t
```

Appends the specified layer data to the `prRoutingLayers` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techGetPrRoutingLayers

```
techGetPrRoutingLayers(
     d_techFileID
     )
     => l_routingLayers/nil
```

Description

Returns a list of the layer and direction data defined in the `prRoutingLayers` subclass of the specified technology file. The `prRoutingLayers` subclass is located in the Place and Route Rules class; it lists the layers in the order of placement. Layers closest to the substrate are listed first.

For more information about the `prRoutingLayers` subclass of the technology file, refer to "prRoutingLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_routingLayers	A list of the layer and usage data defined in the specified technology file. The list has the following syntax: `( ( lt_layer [t_direction] ) ... )` *lt_layer* is the layer-purpose pair or layer name. *t_direction* is the direction, if any, assigned to the layer.
`nil`	The technology file or the `prRoutingLayers` subclass does not exist.

Example

```
techGetPrRoutingLayers( techFileID )
=>(( "poly1" "halfRoute" )
( "metal1" "horizontal" )
( ("metal1" "drawing2") "vertical" )
( "metal2" "vertical" )
)
```

Returns the place-and-route layer data defined by the `prRoutingLayers` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techGetPrRoutingDirection

```
techGetPrRoutingDirection(
     d_techFileID
     ltx_layer
     )
     => t_direction/nil
```

Description

Returns the direction data defined for the specified layer in the `prRoutingLayers` subclass of the specified technology file. The `prRoutingLayers` subclass is located in the Place and Route Rules class; it lists the layers in the order of placement. Layers closest to the substrate are listed first.

For more information about the `prRoutingLayers` subclass of the technology file, refer to "prRoutingLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
ltx_layer	The layer for which you want to get the usage. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

t_direction	The direction assigned to the specified layer.
`nil`	The technology file does not exist, the `prRoutingLayers` subclass does not exist, or the layer is not assigned a routing direction in the `prRoutingLayers` subclass.

Example

```
techGetPrRoutingDirection( techFileID "poly1" )
=> halfRoute
```

Returns the usage of the `poly1` layer defined by the `prRoutingLayers` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techIsPrRoutingLayer

```
techIsPrRoutingLayer(
     d_techFileID
     ltx_layer
     )
     => t/nil
```

Description

Indicates whether the specified layer is listed in the `prRoutingLayers` subclass of the technology file. The `prRoutingLayers` subclass is located in the Place and Route Rules class; it lists the layers in the order of placement. Layers closest to the substrate are listed first.

For more information about the `prRoutingLayers` subclass of the technology file, refer to "prRoutingLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`ltx_layer`	The layer to check. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

`t`	The specified layer is listed in the `prRoutingLayers` subclass in the specified technology file.
`nil`	The technology file does not exist, hte `prRoutingLayers` subclass does not exist, or the layer is not listed in the `prRoutingLayers` subclass.

Example

```
techIsPrRoutingLayer( techFileID "poly1" )
=> t
```

The `poly1` layer is listed in the `prRoutingLayers` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techSetPrViaTypes

```
techSetPrViaTypes(
     d_techFileID
     l_viaTypes
     )
     => t/nil
```

Description

Creates a `prViaTypes` subclass of the specified technology file. The `prViaTypes` subclass is located in the Place and Route Rules class; it lists the devices that the place-and-route tools use as vias and the rules that apply to them. If a `prViaTypes` subclass already exists, this function deletes it and replaces it with the specified data.

For more information about the `prViaTypes` subclass of the technology file, refer to "prViaTypes" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
l_viaTypes	A list of the devices and the optional via type keywords you want to define. The list has the following syntax: `list( list( list( t_cell t_view) [t_type]) ... )` *t_cell* is the cell name of the device. *t_view* is the view name of the device. *t_type* is the keyword that indicates how you want the place-and-route tools to use the device. Valid Values: `default`, the names of rules defined in `prNonDefaultRules` (Preview Gate Ensemble and Preview Silicon Ensemble only)

Return Values

`t`	The `prViaTypes` subclass in the specified technology file was created or replaced.
`nil`	The technology file does not exist.

Example

```
techSetPrViaTypes( techFileID list(
( list(M2_M1    symbolic) "default" )
( list(M3_M2    symbolic) "default" )
( list(ND1M2_M1 symbolic) "NDrule1" )
( list(ND1M3_M2 symbolic) "NDrule1" )
( list(ND2M2_M1 symbolic) "NDrule2" )
( list(ND2M3_M2 symbolic) "NDrule2" )
)
)
=> t
```

Creates a `prViaTypes` subclass with the specified layer and type pairs in the technology file identified by `techFileID`.

techSetPrViaType

```
techSetPrViaType(
     d_techFileID
     l_viaDevice
     t_viaType
     )
     => t/nil
```

Description

Updates the via type of the specified device in the `prViaTypes` subclass of the specified technology file. The `prViaTypes` subclass is located in the Place and Route Rules class; it lists the devices the place-and-route tools use as vias and the rules that apply to them. If the device is not listed in the `prViaTypes` subclass, this function appends the device and via type. If the `prViaTypes` subclass does not exist, this function creates it with the specified data.

For more information about the `prViaTypes` subclass of the technology file, refer to "prViaTypes" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
l_viaDevice	A list containing the cell and view names of the device. The list has the following syntax: `list( t_cell t_view )` *t_cell* is the cell name of the device. *t_view* is the view name of the device.
t_viaType	The keyword that indicates how you want the place-and-route tools to use the device. Valid Values: `default`, the names of rules defined in `prNonDefaultRules()` (Preview Gate Ensemble and Preview Silicon Ensemble only)

Return Values

`t`	The data was added to the `prViaTypes` subclass in the specified technology file.
`nil`	The technology file does not exist.

Example

```
techSetPrViaType( techFileID list("M2_M1" "symbolic") "default" )
=> t
```

Appends the specified via device data to the `prViaTypes` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techGetPrViaTypes

```
techGetPrViaTypes(
     d_techFileID
     )
     => l_viaTypes/nil
```

Description

Returns a list of the vias defined in the `prViaTypes` subclass of the specified technology file. The `prViaTypes` subclass is located in the Place and Route Rules class; it lists the devices that the place-and-route tools use as vias and the rules that apply to them.

For more information about the `prViaTypes` subclass of the technology file, refer to "prViaTypes" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_viaTypes	A list containing the cell and view names of the device and their assigned via types. The list has the following syntax: `list( list( list( t_cell t_view ) t_viaType) ... )` *t_cell* is the cell name of the device. *t_view* is the view name of the device. *t_viaType* is the via type assigned to the device.
`nil`	The technology file or the `prViaTypes` subclass does not exist.

Example

```
techGetPrViaTypes( techFileID )
=> ( (M2_M1    symbolic) "default" )
( (M3_M2    symbolic) "default" )
( (ND1M2_M1 symbolic) "NDrule1" )
( (ND1M3_M2 symbolic) "NDrule1" )
( (ND2M2_M1 symbolic) "NDrule2" )
( (ND2M3_M2 symbolic) "NDrule2" )
```

Returns the place-and-route via device data defined by the `prViaTypes` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techGetPrViaType

```
techGetPrViaType(
     d_techFileID
     l_viaDevice
     )
     => t_viaType/nil
```

Description

Returns the via type defined for the specified layer in the `prViaTypes` subclass of the specified technology file. The `prViaTypes` subclass is located in the Place and Route Rules class; it lists the devices the place-and-route tools use as vias and the rules that apply to them.

For more information about the `prViaTypes` subclass of the technology file, refer to "prViaTypes" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
l_viaDevice	A list containing the cell and view names of the device for which you want the via type.

Return Values

t_viaType	The via type assigned to the specified device.
`nil`	The technology file does not exist, the `prViaTypes` subclass does not exist, or the device is not assigned a via type in the `prViaTypes` subclass.

Example

```
techGetPrViaType( techFileID list("M2_M1" "symbolic")
=> default
```

Returns the via type assigned the `M2_M1` device in the `prViaTypes` class of the Place and Route Rules class of the technology file identified by `techFileID`.

techIsPrViaDevice

```
techIsPrViaDevice(
     d_techFileID
     l_viaDevice
     )
     => t/nil
```

Description

Indicates whether the specified device is listed in the `prViaTypes` subclass of the technology file. The `prViaTypes` subclass is located in the Place and Route Rules class; it lists the devices the place-and-route tools use as vias and the rules that apply to them.

For more information about the `prViaTypes` subclass of the technology file, refer to "prViaTypes" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`l_viaDevice`	A list containing the cell and view names to check.

Return Values

`t`	The specified device is listed in the `prViaTypes` class of the specified technology file.
`nil`	The technology file does not exist, the `prViaTypes` subclass does not exist, or the device is not listed in the `prViaTypes` subclass.

Example

```
techIsPrViaDevice( techFileID list("M3_M2" "symbolic"))
=> t
```

The `M3_M2` device is listed in the `prViaTypes` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techSetPrStackVias

```
techSetPrStackVias(
     d_techFileID
     l_stackVias
     )
     => t/nil
```

Description

Creates a `prStackVias` subclass of the specified technology file. The `prStackVias` subclass is an optional subclass used by the Preview Gate Ensemble and Preview Silicon Ensemble products. This subclass is located in the Place and Route Rules class; it lists pairs of contact via layers that can be stacked. If a `prStackVias` subclass already exists, this function deletes it and replaces it with the specified data..

For more information about the `prStackVias` subclass of the technology file, refer to "prStackVias" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
l_stackVias	A list of two layers. The list has the following syntax: `list( list( ltx_viaLayer1 ltx_viaLayer2 ) ... )` *ltx_viaLayer1* is one of the layers in the via layer pair. Valid Values: the layer name, the layer number, a list containing the layer name and purpose *ltx_viaLayer2* is the second layer of the pair. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

`t`	The `prStackVias` subclass in the specified technology file was created or replaced.
`nil`	The technology file does not exist.

Example

```
techSetPrStackVias( techFileID
list( "via" "via2" )
)
=> t
```

Creates a `prStackVias` subclass with the specified layer pairs in the technology file identified by `techFileID`.

techSetPrStackVia

```
techSetPrStackVia(
     d_techFileID
     ltx_viaLayer1
     ltx_viaLayer2
     )
     => t/nil
```

Description

Updates the `prStackVias` subclass of the specified technology file with the via layer pair. The `prStackVias` subclass is an optional subclass used by the Preview Gate Ensemble and Preview Silicon Ensemble products. This subclass is located in the Place and Route Rules class; it lists pairs of via layers that can be stacked. If the `prStackVias` subclass does not exist, this function creates it with the specified data.

For more information about the `prStackVias` subclass of the technology file, refer to "prStackVias" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`ltx_viaLayer1`	One of the layers in the via layer pair. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
`ltx_viaLayer2`	The second layer in the via layer pair. Valid Values: same as for *`ltx_viaLayer1`*

Return Values

`t`	The data was added to the `prStackVias` subclass in the specified technology file.
`nil`	The technology file does not exist.

Example

```
techSetPrStackVia( techFileID "via" "via2")
=> t
```

Appends the specified via layer pair to the `prStackVias` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techGetPrStackVias

```
techGetPrStackVias(
     d_techFileID
     )
     => l_stackVias/nil
```

Description

Returns a list of the via layer pairs defined in the `prStackVias` subclass of the specified technology file. The `prStackVias` subclass is an optional subclass used by the Preview Gate Ensemble and Preview Silicon Ensemble products. This subclass is located in the Place and Route Rules class; it lists pairs of via layers that can be stacked.

For more information about the `prStackVias` subclass of the technology file, refer to "prStackVias" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_stackVias	A list containing the cell and view names of the device and their assigned via types. The list has the following syntax: `list( list( lt_layer1 lt_layer2 ) ... )` *lt_layer1* is the first layer. *lt_layer2* is the second layer.
`nil`	The technology file or the `prStackVias` subclass does not exist.

Example

```
techGetPrStackVias( techFileID )
=> ( "via" "via2" )
```

Returns the via layer pairs defined in the `prStackVias` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techIsPrStackVia

```
techIsPrStackVia(
     d_techFileID
     ltx_viaLayer1
     ltx_viaLayer2
     )
     => t/nil
```

Description

Indicates whether the specified via layer pair is listed in the `prStackVias` subclass of the technology file. The `prStackVias` subclass is an optional subclass used by the Preview Gate Ensemble and Preview Silicon Ensemble products. This subclass is located in the Place and Route Rules class; it lists pairs of via layers that can be stacked.

For more information about the `prStackVias` subclass of the technology file, refer to "prStackVias" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`ltx_viaLayer1`	One of the layers in the via layer pair. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
`ltx_viaLayer2`	The second layer in the via layer pair. Valid Values: same as for *`ltx_viaLayer1`*

Return Values

`t`	The specified via layer pair is listed in the `prStackVias` subclass in the specified technology file.
`nil`	The technology file does not exist, the `prStackVias` subclass does not exist, or the layer is not listed in the `prStackVias` subclass.

Example

```
techIsPrStackVia( techFileID "via" "via2")
=> t
```

The `via`/`via2` layer pair is listed in the `prStackVias` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techSetPrMaxStackVia

```
techSetPrMaxStackVia(
     d_techFileID
     x_value
     [ t_bottomLayer t_topLayer ]
     )
     => t/nil
```

Description

Updates the `prMaxStackVias` subclass of the specified technology file with one set of maximum stacked vias data. The `prMaxStackVias` subclass is an optional subclass located in the Place and Route Rules class; it defines the maximum number of stacked vias allowed within a specified layer range or ranges. If the `prMaxStackVias` subclass does not exist, this function creates it with the specified data.

For more information about the `prMaxStackVias` subclass of the technology file, refer to "prMaxStackVias" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`x_value`	The maximum number of stacked vias allowed within the layer range. Valid values: Any positive integer
`lxt_bottomLayer`	The bottom layer of the layer range. Valid values: Any routing layer
`ltx_topLayer`	The top layer of the layer range. Valid values: Any routing layer

Return Values

`t`	The `prMaxStackVias` subclass in the specified technology file was updated or created.
`nil`	The technology file does not exist.

Example

```
techSetPrMaxStackVia( techFileID
4  metal1  metal5
)
=> t
```

Updates or creates the `prMaxStackVias` subclass in the technology file identified by `techFileID` to allow stacking of up to four vias using `metal1` through `metal5` routing layers.

techSetPrMaxStackVias

```
techSetPrStackVias(
     d_techFileID
     l_stackVias
     )
     => t/nil
```

Description

Replaces the `prMaxStackVias` subclass of the specified technology file with one or more sets of maximum stacked vias data. The `prMaxStackVias` subclass is an optional subclass located in the Place and Route Rules class; it defines the maximum number of stacked vias allowed within a specified layer range or ranges. If the `prMaxStackVias` subclass does not exist, this function creates it with the specified data.

For more information about the `prStackVias` subclass of the technology file, refer to "prMaxStackVias" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

l_stackVias	A list containing the maximum stacked via number and the layer range. The list has the following syntax:
	`list(`
	`( x_value [ lxt_bottomLayer lxt_topLayer ] )`
	`...`
	`)`
	
	x_value is the maximum number of stacked vias allowed within the layer range.
	
	ltx_bottomLayer is the bottom layer of the layer range.
	Valid Values: the layer name, the layer number, a list containing the layer name and purpose
	
	ltx_topLayer is the top layer of the layer range.
	Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

`t`	The `prMaxStackVias` subclass in the specified technology file was updated or created.
`nil`	The technology file does not exist.

Example

```
techSetPrMaxStackVias( techFileID
( 4  metal1  metal5 )
( 2  metal5  metal7 )
)
=> t
```

Updates or creates the `prMaxStackVias` subclass in the technology file identified by `techFileID` to allow stacking of up to four vias using `metal1` through `metal5` routing layers and to allow stacking of up to two vias using `metal5` through `metal7` routing layers..

techSetPrMastersliceLayers

```
techSetPrMastersliceLayers(
     d_techFileID
     l_msLayers
     )
     => t/nil
```

Description

Creates a `prMastersliceLayers` subclass of the specified technology file. The `prMastersliceLayers` subclass is an optional subclass used by the Preview Gate Ensemble and Preview Silicon Ensemble products. This subclass is located in the Place and Route Rules class; it lists masterslice layers. If a `prMastersliceLayers` subclass already exists, this function deletes it and replaces it with the specified data.

For more information about the `prMastersliceLayers` subclass of the technology file, refer to "prMastersliceLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
l_msLayers	A list of the layers you want to define as masterslice layers. The list has the following syntax: `list( ltx_msLayer ... )` *ltx_msLayer* is a layer to specify as a masterslice layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

`t`	The `prMastersliceLayers` subclass in the specified technology file was created or replaced.
`nil`	The technology file does not exist.

Example

```
techSetPrMastersliceLayers( techFileID
list( "diff" "poly" )
)
=> t
```

Creates a `prMastersliceLayers` subclass with the specified layers in the technology file identified by `techFileID`.

techSetPrMastersliceLayer

```
techSetPrMastersliceLayer(
     d_techFileID
     ltx_msLayer
     )
     => t/nil
```

Description

Updates the `prMastersliceLayers` subclass of the specified technology file with the specified layer. The `prMastersliceLayers` subclass is an optional subclass used by the Preview Gate Ensemble and Preview Silicon Ensemble products. This subclass is located in the Place and Route Rules class; it lists masterslice layers. If the `prMastersliceLayers` subclass does not exist, this function creates it with the specified layer.

For more information about the `prMastersliceLayers` subclass of the technology file, refer to "prMastersliceLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`ltx_msLayer`	A layer to specify as a masterslice layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

`t`	The data was added to the `prMastersliceLayers` subclass in the specified technology file.
`nil`	The technology file does not exist.

Example

```
techSetPrMastersliceLayer( techFileID "poly" )
=> t
```

Appends the `poly` layer to the `prMastersliceLayers` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techGetPrMastersliceLayers

```
techGetPrMastersliceLayers(
     d_techFileID
     )
     => l_msLayers/nil
```

Description

Returns a list of the masterslice layers defined in the `prMastersliceLayers` subclass of the specified technology file. The `prMastersliceLayers` subclass is an optional subclass used by the Preview Gate Ensemble and Preview Silicon Ensemble products. This subclass is located in the Place and Route Rules class; it lists masterslice layers.

For more information about the `prMastersliceLayers` subclass of the technology file, refer to "prMastersliceLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_msLayers	A list containing the cell and view names of the device and their assigned via types. The list has the following syntax: `list( lt_mastersliceLayer ... )` *lt_mastersliceLayer* is a layer specified as a masterslice layer.
`nil`	The technology file or the `prMastersliceLayers` subclass does not exist.

Example

```
techGetPrMastersliceLayers( techFileID )
=> ( "diff" "poly" )
```

Returns the layers defined by the `prMastersliceLayers` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techIsPrMastersliceLayer

```
techIsPrMastersliceLayer(
     d_techFileID
     ltx_msLayer
     )
     => t/nil
```

Description

Indicates whether the specified layer is listed in the `prMastersliceLayers` subclass of the technology file. The `prMastersliceLayers` subclass is an optional subclass used by the Preview Gate Ensemble and Preview Silicon Ensemble products. This subclass is located in the Place and Route Rules class; it lists masterslice layers.

For more information about the `prMastersliceLayers` subclass of the technology file, refer to "prMastersliceLayers" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`ltx_msLayer`	The layer to check. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

`t`	The specified layer is listed in the `prMastersliceLayers` subclass in the specified technology file.
`nil`	The technology file does not exist, the `prMastersliceLayers` subclass does not exist, or the layer is not listed in the `prMastersliceLayers` subclass.

Example

```
techIsPrMastersliceLayer( techFileID "diff" )
=> t
```

The `diff` layer is listed in the `prMastersliceLayers` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techSetPrViaRule

```
techSetPrViaRule(
     d_techFileID
     t_ruleName
     (t_viaDeviceName)
     ltx_layer1
     t_direction1
     l_params1
     ltx_layer2
     t_direction2
     l_params2
     )
     => t/nil
```

Description

Updates the parameters for the specified via rule in the `prViaRules` subclass of the specified technology file. The `prViaRules` subclass is located in the Place and Route Rules class; it is used by the Preview Gate Ensemble and Preview Silicon Ensemble special router to insert vias while routing power and clock. If the rule is not listed in the `prViaRules` subclass, this function appends the rule with the specified parameters. If the `prViaRules` subclass does not exist, this function creates it with the specified data.

For more information about the `prViaRules` subclass of the technology file, refer to "prViaRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_ruleName	The name of the rule. Valid Values: any string (via rule names must be unique)
t_viaDeviceName	The name of the via device. Valid Values: any device listed in "ruleContactDevice" in the *Technology File and Display Resource File ASCII Syntax Reference Manual* in the Devices class of the technology file and enclosed in parentheses
ltx_layer1	The routing layer for the top of the via. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

t_direction1 — The routing direction of the first layer.
Valid Values: `horizontal`, `vertical`

l_params1 — A list of parameters for the first routing layer. The syntax of the list is as follows:

```
list( n_minWidth n_maxWidth n_overhang
      n_metalOverhang )
```

n_minWidth is the minimum width allowed for the wire, in user units.

n_maxWidth is the maximum width allowed for the wire, in user units.

n_overhang is the minimum spacing between the contact cut and the outer edge of the via.

n_metalOverhang is the minimum overhang of the via to the wire.

ltx_layer2 — The routing layer for the bottom of the layer.
Valid Values: the layer name, the layer number, a list containing the layer name and purpose

t_direction2 — The routing direction of the second layer.
Valid Values: the direction perpendicular to *t_direction1*: `vertical`, `horizontal`

l_params2 — A list of parameters for the second routing layer. The syntax is the same as for *l_params1*.

Return Values

`t` — The rule was updated or added to the `prViaRules` subclass in the specified technology file.

`nil` — The technology file does not exist.

Example

```
techSetPrViaRule( techFileID "viaSP21"   (M2_M1)
"metal1" "vertical"   ( .6  1.8   _NA_      .6 )
"metal2" "horizontal" ( .6  1.8   _NA_      .6 )  ))
=> t
```

Appends the specified via rule data to the `prViaRules` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techGetPrViaRules

```
techGetPrViaRules(
     d_techFileID
     )
     => l_viaRules/nil
```

Description

Returns the via rules defined in the `prViaRules` subclass of the specified technology file. The `prViaRules` subclass is located in the Place and Route Rules class; it is used by the Preview Gate Ensemble and Preview Silicon Ensemble special router to insert vias while routing power and clock.

For more information about the `prViaRules` subclass of the technology file, refer to "prViaRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_viaRules — A list of the rules defined in the `prViaRules` subclass in the Place and Route Rules class of the specified technology file. The list has the following syntax:

```
( ( t_ruleName t_device lt_layer1 t_dir1
( n_minWidth n_maxWidth n_overhang n_metalOverHang )
lt_layer2 t_dir2
( n_minWidth n_maxWidth n_overhang n_metalOverHang )
)
... )
```

t_ruleName is the name of the rule.

t_device is the name of the via device.

lt_layer1 specifies the routing for the top and bottom of the via.

lt_layer2 specifies the routing for the top and bottom of the via.

t_dir1 is the routing direction of the corresponding layer.

t_dir2 is the routing direction of the corresponding layer.

n_minWidth is the minimum width allowed for the wire, in user units.

n_maxWidth is the maximum width allowed for the wire, in user units.

n_overhang is the minimum spacing between the contact cut and the outer edge of the via.

n_metalOverhang is the minimum overhang of the via to the wire.

`nil` The technology file or `prViaRules` subclass does not exist.

Example

```
techGetPrViaRules( techFileID )
=> (
( "viaSP21" (M2_M1)
"metal1" "vertical"   ( .6  1.8   _NA_      .6 )
"metal2" "horizontal" ( .6  1.8   _NA_      .6 ) )
( "viaSP32" (M3_M2)
"metal2" "horizontal" ( .6  1.8   _NA_      .6 )
"metal3" "vertical"   ( .6  1.8   _NA_      .6 ) )
)
```

Returns the rules defined in the `prViaRules` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techGetPrViaParams

```
techGetPrViaParams(
     d_techFileID
     t_ruleName
     )
     => l_viaParams/nil
```

Description

Returns the parameters of the specified via rule defined in the `prViaRules` subclass of the specified technology file. The `prViaRules` subclass is located in the Place and Route Rules class; it is used by the Preview Gate Ensemble and Preview Silicon Ensemble special router to insert vias while routing power and clock. If the technology file or `prViaRules` subclass does not exist, this function returns `nil`.

For more information about the `prViaRules` subclass of the technology file, refer to "prViaRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_ruleName	The name of the rule.

Return Values

l_viaParams	A list of the parameters defined in the `prViaRules` subclass for the specified via rule in the Place and Route Rules class of the specified technology file. The list has the following syntax:
	`( t_device lt_layer1 t_dir1`
	`( n_minWidth n_maxWidth n_overhang n_metalOverHang )`
	`lt_layer2 t_dir2`
	`( n_minWidth n_maxWidth n_overhang n_metalOverHang )`
	`)`
	
	t_device is the name of the via device.
	
	lt_layer1 is the routing and masterslice layers.
	
	lt_layer2 specifies the routing and masterslice layers.

t_dir1 is the routing direction of the corresponding layer.

t_dir2 is the routing direction of the corresponding layer.

n_minWidth is the minimum width allowed for the wire, in user units.

n_maxWidth is the maximum width allowed for the wire, in user units.

n_overhang is the minimum spacing between the contact cut and the outer edge of the via.

n_metalOverhang is the minimum overhang of the via to the wire.

`nil` The technology file does not exist, the `prViaRules` subclass does not exist, or the specified rule is not defined in the `prViaRules` subclass.

Example

```
techGetPrViaParams( techFileID "viaSP21" )
=> ( (M2_M1)
"metal1" "vertical"   ( .6  1.8   _NA_      .6 )
"metal2" "horizontal" ( .6  1.8   _NA_      .6 )  ))
```

Returns the specified via rule data from the `prViaRules` subclass of the technology file identified by `techFileID`.

techSetPrViaProp

```
techSetPrViaProp(
     d_techFileID
     t_ruleName
     t_propname
     g_value
     )
     => t/nil
```

Description

Sets the property of the specified via rule defined in the `prViaRules` subclass of the specified technology file. The `prViaRules` subclass is located in the Place and Route Rules class; it is used by the Preview Gate Ensemble and Preview Silicon Ensemble special router to insert vias while routing power and clock. If the technology file or `prViaRules` subclass does not exist, this function returns `nil`.

For more information about the `prViaRules` subclass of the technology file, refer to "prViaRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_ruleName	The name of the rule.
t_propName	The name of the property you want to set.
g_value	The value assigned to the poperty.

Return Values

`t`	The property was set.
`nil`	The technology file does not exist, the `prViaRules` subclass does not exist, or the specified rule is not defined in the `prViaRules` subclass.

techGetPrViaProps

```
techGetPrViaProps(
     d_techFileID
     t_ruleName
     )
     => l_viaProps/nil
```

Description

Returns the properties of the specified via rule defined in the `prViaRules` subclass of the specified technology file. The `prViaRules` subclass is located in the Place and Route Rules class; it is used by the Preview Gate Ensemble and Preview Silicon Ensemble special router to insert vias while routing power and clock. If the technology file or `prViaRules` subclass does not exist, this function returns `nil`.

For more information about the `prViaRules` subclass of the technology file, refer to "prViaRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_ruleName	The name of the rule.

Return Values

l_viaProps	A list of the properties defined for the via and the values you want to use for this via. The syntax is as follows: `( ( t_propName g_propValue ) ... )` *t_propName* is the name of the property you want to set. *t_propValue* is the value assigned to the property.
`nil`	The technology file does not exist, the `prViaRules` subclass does not exist, or the specified rule is not defined in the `prViaRules` subclass.

techSetPrGenViaRule

```
techSetPrGenViaRule(
     d_techFileID
     t_ruleName
     l_genViaRule
     )
     => t/nil
```

Description

Updates the parameters for the generated via rule specified in the `prGenViaRules` subclass of the specified technology file. The `prGenViaRules` subclass is located in the Place and Route Rules class; it is used by the Preview Gate Ensemble and Preview Silicon Ensemble special router to generate vias while routing power and clock. If the rule is not listed in the `prGenViaRules` subclass, this function appends the rule with the specified parameters. If the `prGenViaRules` subclass does not exist, this function creates it with the specified data.

For more information about the `prGenViaRules` subclass of the technology file, refer to "prGenViaRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_ruleName	The name of the rule. Valid Values: any string (via rule names must be unique)
l_genViaRule	A list containing the rule parameters. The list has the following syntax: *ltx_viaLayer* *(n_width n_length n_xPitch n_yPitch n_resistance)* *ltx_layer1 t_dir1* *(n_minWidth n_maxWidth n_overhang n_metalOverHang)* *ltx_layer2 t_dir2* *(n_minWidth n_maxWidth n_overhang n_metalOverHang)* *ltx_viaLayer* is the via layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose *n_width* is the width of the via layer.

n_length is the length of the via layer.

n_xPitch is the x pitch of the via layer.

n_yPitch is the y pitch of the via layer.

n_resistance is the resistance value.

ltx_layer1 is the routing layers for the top and bottom of the via.
Valid Values: the layer name, the layer number, a list containing the layer name and purpose

ltx_layer2 is the routing layers for the top and bottom of the via.
Valid Values: the layer name, the layer number, a list containing the layer name and purpose

t_dir1 is the routing direction of the top and bottom layers.
Valid Values: `horizontal`, `vertical` (the layers must be assigned perpendicular directions)

t_dir2 is the routing direction of the top and bottom layers.
Valid Values: `horizontal`, `vertical` (the layers must be assigned perpendicular directions)

n_minWidth is the minimum width allowed for the wire, in user units.

n_maxWidth is the maximum width allowed for the wire, in user units.

n_overhang is the minimum spacing between the contact cut and the outer edge of the via.

n_metalOverhang is the minimum overhang of the via to the wire.

Return Values

`t`	The rule was updated or added to the `prGenViaRules` subclass in the specified technology file.

`nil` The technology file does not exist.

Example

```
techSetPrGenViaRule( techFileID (viagen21
via       ( .6 .6  1.2    1.2    _NA_ )
metal1   "horizontal" ( .6  20.0  .6       .6 )
metal2   "vertical"   ( .6  20.0  .6       .6 )  )
=> t
```

Appends the specified via rule data to the `prGenViaRules` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techGetPrGenViaRules

```
techGetPrGenViaRules(
     d_techFileID
     )
     => l_genViaRules/nil
```

Description

Returns the generated via rules defined in the `prGenViaRules` subclass of the specified technology file. The `prGenViaRules` subclass is located in the Place and Route Rules class; it is used by the Preview Gate Ensemble and Preview Silicon Ensemble special router to generate vias while routing power and clock.

For more information about the `prGenViaRules` subclass of the technology file, refer to "prGenViaRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_genViaRules	A list of the rules defined in the `prGenViaRules` subclass in the Place and Route Rules class of the specified technology file. The list has the following syntax:
	`( ( t_ruleName lt_viaLayer`
	`( n_width n_length n_xPitch n_yPitch n_resistance )`
	`lt_layer1 t_dir1`
	`( n_minWidth n_maxWidth n_overhang n_metalOverHang )`
	`lt_layer2 t_dir2`
	`( n_minWidth n_maxWidth n_overhang n_metalOverHang )`
	`)`
	`... )`
	
	t_ruleName is the name of the rule.
	
	lt_viaLayer is the via layer.
	
	n_width is the width of the via layer.
	
	n_length is the length of the via layer.

n_xPitch is the x pitch of the via layer.

n_yPitch is the y pitch of the via layer.

n_resistance is the resistance value.

lt_layer1 specifies the routing layers for the top and bottom of the via.

lt_layer2 specifies the routing layers for the top and bottom of the via.

t_dir1 is the routing direction of the top and bottom layers.

t_dir2 is the routing direction of the top and bottom layers.

n_minWidth is the minimum width allowed for the wire, in user units.

n_maxWidth is the maximum width allowed for the wire, in user units.

n_overhang is the minimum spacing between the contact cut and the outer edge of the via.

n_metalOverhang is the minimum overhang of the via to the wire.

`nil` The technology file or the `prGenViaRules` subclass does not exist.

Example

```
techGetPrGenViaRules( techFileID )
=> (
(viagen21 via ( .6 .6  1.2 1.2 _NA_ )
 metal1 "horizontal" ( .6 20.0 .6 .6 )
 metal2 "vertical"   ( .6 20.0 .6 .6 )  )
(viagen32 via2 ( .6 .6 1.2 1.2 _NA_ )
 metal2 "vertical" ( .6 20.0 .6 .6 )
 metal3 "horizontal" ( .6 20.0 .6 .6 )  )
)
```

Returns the rules defined in the `prGenViaRules` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techGetPrGenViaParams

```
techGetPrGenViaParams(
     d_techFileID
     t_ruleName
     )
     => l_genViaParams/nil
```

Description

Returns the parameters of the specified generated via rule defined in the `prGenViaRules` subclass of the specified technology file. The `prGenViaRules` subclass is located in the Place and Route Rules class; it is used by the Preview Gate Ensemble and Preview Silicon Ensemble special router to generate vias while routing power and clock.

For more information about the `prGenViaRules` subclass of the technology file, refer to "prGenViaRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_ruleName	The name of the rule.

Return Values

l_genViaParams	A list containing the data defined for the specified rule in the `prGenViaRules` subclass in the Place and Route Rules class of the specified technology file. The list has the following syntax:

```
( lt_viaLayer
( n_width n_length n_xPitch n_yPitch n_resistance )
lt_layer1 t_dir1
( n_minWidth n_maxWidth n_overhang n_metalOverHang )
lt_layer2 t_dir2
( n_minWidth n_maxWidth n_overhang n_metalOverHang )
)
```

lt_viaLayer is the via layer.

n_width is the width of the via layer.

n_length is the length of the via layer.

n_xPitch is the x pitch of the via layer.

n_yPitch is the y pitch of the via layer.

n_resistance is the resistance value.

lt_layer1 specifies the routing layers for the top and bottom of the via.

lt_layer2 specifies the routing layers for the top and bottom of the via.

t_dir1 is the routing direction of the top and bottom layers.

t_dir2 is the routing direction of the top and bottom layers.

n_minWidth is the minimum width allowed for the wire, in user units.

n_maxWidth is the maximum width allowed for the wire, in user units.

n_overHang is the minimum spacing between the contact cut and the outer edge of the via.

n_metalOverhang is the minimum overhang of the via to the wire.

`nil` The technology file does not exist, the `prGenViaRules` subclass does not exist, or the specified rule is not defined in the `prGenViaRules` subclass.

Example

```
techGetPrGenViaParams( techFileID "viagen21" )
=> ( via ( .6 .6  1.2 1.2 _NA_ )
 metal1 "horizontal" ( .6 20.0 .6 .6 )
 metal2 "vertical"   ( .6 20.0 .6 .6 )  )
```

Returns the parameters of the specified rule defined in the `prGenViaRules` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techSetPrGenViaProp

```
techSetPrGenViaProp(
     d_techFileId
     t_ruleName
     t_propName
     g_value
     )
     => t/nil
```

Description

Sets the property of the specified file generated via rule defined in the `prGenViaRules` subclass of the specified technology file. The `prGenViaRules` subclass is located in the Place and Route Rules class; it is used by the Preview Gate Ensemble and Preview Silicon Ensemble special router to generate vias while routing power and clock.

For more information about the `prGenViaRules` subclass of the technology file, refer to "prGenViaRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_ruleName	The name of the rule.
t_propName	The name of the property you want to set.
g_value	The value assigned to the property.

Return Values

`t`	The property was set.
`nil`	The technology file does not exist, the `prGenViaRules` subclass does not exist, or the specified rule is not defined in the `prGenViaRules` subclass.

techGetPrGenViaProps

```
techGetPrGenViaProps(
     d_techFileId
     t_ruleName
     )
     => l_genViaProps/nill
```

Description

Returns the properties of the specified file generated via rule defined in the `prGenViaRules` subclass of the specified technology file. The `prGenViaRules` subclass is located in the Place and Route Rules class; it is used by the Preview Gate Ensemble and Preview Silicon Ensemble special router to generate vias while routing power and clock.

For more information about the `prGenViaRules` subclass of the technology file, refer to "prGenViaRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_ruleName	The name of the rule.

Return Values

l_genViaProps	A list of the properties defined for the via and the values you want to use for this via. The syntax is as follows: `( ( t_propName g_propValue ) ... )` *t_propName* is the name of the property you want to set. *t_propValue* is the value assigned to the property.
`nil`	The technology file does not exist, the `prGenViaRules` subclass does not exist, or the specified rule is not defined in the `prGenViaRules` subclass.

techSetPrNonDefaultRule

```
techSetPrNonDefaultRule(
     d_techFileID
     t_ruleName
     l_nonDefaultRule
     )
     => t/nil
```

Description

Updates the parameters for the specified nondefault rule in the `prNonDefaultRules` subclass of the specified technology file. The `prNonDefaultRules` subclass is located in the Place and Route Rules class; it lists the rules that Preview Silicon Ensemble and Preview Gate Ensemble use when you route with nondefault wire widths. If the rule is not listed in the `prNonDefaultRules` subclass, this function appends the rule with the specified parameters. If the `prNonDefaultRules` subclass does not exist, this function creates it with the specified data.

For more information about the `prNonDefaultRules` subclass of the technology file, refer to "prNonDefaultRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_ruleName	The name of the rule. Valid Values: any string (rule names must be unique)

l_nonDefaultRule	A list containing the rule parameters. The list has the following syntax:
	`list(`
	`( ( ltx_layer n_width n_spacing [n_notch] ) ... )`
	`( t_viaDevice ... )`
	`( ( ltx_viaLayer1 ltx_viaLayer2 n_minSpace g_stack )`
	`... )`
	`)`
	
	ltx_layer is the routing layer
	Valid Values: the layer name, the layer number, a list containing the layer name and purpose
	
	n_width is the width of the layer.

n_spacing is the minimum spacing allowed for the layer.

n_notch is the notch spacing allowed for the layer.

t_viaDevice is the name of the via device.
Valid Values: any device listed in the ruleContactDevice subclass in the Devices class of the technology file

ltx_viaLayer1-2 specifies the routing layers for the top and bottom of the via.
Valid Values: the layer name, the layer number, a list containing the layer name and purpose

n_minSpace is the minimum spacing allowed between via layers.

g_stack indicates whether you can stack the via layers.
Valid Values: t (can be stacked), nil (cannot be stacked)

Return Values

t	The rule was updated or added to the prNonDefaultRules subclass in the specified technology file.
nil	The technology file does not exist.

Example

```
techSetPrNonDefaultRule( techFileID "NDrule1" list(
((metal1 2.4 .8 .6   )
 (metal2 2.4 .8 .6   )
 (metal3 2.4 .8 _NA_ ) )
(ND1M2_M1 ND1M3_M2 )
( (via via2 .4 t) )
)
=> t
```

Appends the specified nondefault rule data to the prNonDefaultRules subclass in the Place and Route Rules class of the technology file identified by techFileID.

techGetPrNonDefaultRules

```
techGetPrNonDefaultRules(
     d_techFileID
     )
     => l_nonDefaultRules/nil
```

Description

Returns the nondefault rules defined in the `prNonDefaultRules` subclass of the specified technology file. The `prNonDefaultRules` subclass is located in the Place and Route Rules class; it lists the rules that Preview Silicon Ensemble and Preview Gate Ensemble use when you route with nondefault wire widths.

For more information about the `prNonDefaultRules` subclass of the technology file, refer to "prNonDefaultRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_nonDefaultRules	A list of the rules defined in the `prNonDefaultRules` subclass in the Place and Route Rules class of the specified technology file. The list has the following syntax:
	`( (`
	`( ( lt_layer n_width n_spacing [n_notch] ) ... )`
	`( t_viaDevice ... )`
	`( ( lt_viaLayer1 lt_viaLayer2 n_minSpace g_stack )`
	`... ) ) ... )`
	
	lt_layer is the routing layer.
	
	n_width is the width of the layer.
	
	n_spacing is the minimum spacing allowed for the layer.
	
	n_notch is the notch spacing allowed for the layer.
	
	t_viaDevice is the name of the via device.

lt_viaLayer1 specifies the routing layers for the top and bottom of the via.

lt_viaLayer2 specifies the routing layers for the top and bottom of the via.

n_minSpace is the minimum spacing allowed between via layers.

g_stack indicates whether you can stack the via layers.

`nil` The technology file or `prNonDefaultRules` subclass does not exist.

Example

```
techGetPrNonDefaultRules( techFileID )
=> (
("NDrule1"
((metal1 2.4 .8 .6   )
 (metal2 2.4 .8 .6   )
 (metal3 2.4 .8 _NA_ ) )
((ND1M2_M1 ND1M3_M2 )
( (via via2 .4 t)
("NDrule2"
((metal1    3.0   1.0     .6   )
 (metal2    3.0   1.0     .6   )
 (metal3    3.0   1.0     _NA_ ) )
((ND2M2_M1 ND2M3_M2 )
)
```

Returns the rules defined in the `prNonDefaultRules` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techGetPrNonDefaultParams

```
techGetPrNonDefaultParams(
     d_techFileID
     t_ruleName
     )
     => l_nonDefaultParams/nil
```

Description

Returns the parameters of the specified nondefault rule defined in the `prNonDefaultRules` subclass of the specified technology file. The `prNonDefaultRules` subclass is located in the Place and Route Rules class; it lists the rules that Preview Silicon Ensemble and Preview Gate Ensemble use when you route with nondefault wire widths.

For more information about the `prNonDefaultRules` subclass of the technology file, refer to "prNonDefaultRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_ruleName	The name of the rule.

Return Values

l_nonDefaultParams	A list containing the data defined for the specified rule in the `prNonDefaultRules` subclass in the Place and Route Rules class of the specified technology file. The list has the following syntax:
	`( (`
	`( ( lt_layer n_width n_spacing [n_notch] ) ... )`
	`( t_viaDevice ... )`
	`( ( lt_viaLayer1 lt_viaLayer2 n_minSpace g_stack )`
	`... ) ) ...)`
	
	lt_layer is the routing layer.
	
	n_width is the width of the layer.
	
	n_spacing is the minimum spacing allowed for the layer.

n_notch is the notch spacing allowed for the layer.

t_viaDevice is the name of the via device.

lt_viaLayer1 specifies the routing layers for the top and bottom of the via.

lt_viaLayer2 specifies the routing layers for the top and bottom of the via.

n_minSpace is the minimum spacing allowed between via layers.

g_stack indicates whether you can stack the via layers.

`nil` The technology file does not exist, the `prNonDefaultRules` subclass does not exist, or the rule is not defined in the `prNonDefaultRules` subclass.

Example

```
techGetPrNonDefaultParams( techFileID "NDrule1" )
=> (
((metal1 2.4 .8 .6   )
 (metal2 2.4 .8 .6   )
 (metal3 2.4 .8 _NA_ ) )
((ND1M2_M1 .8 )
 (ND1M3_M2 .8 ) )
( (via via2) ) )
```

Returns the parameters for the specified rule defined in the `prNonDefaultRules` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techSetPrNonDefaultProp

```
techSetPrNonDefaultProp(
     d_techId
     t_ruleName
     t_propName
     g_value
     )
     => t/nil
```

Description

Sets the property of the specified nondefault rule defined in the `prNonDefaultRules` subclass of the specified technology file. The `prNonDefaultRules` subclass is located in the Place and Route Rules class; it lists the rules that Preview Silicon Ensemble and Preview Gate Ensemble use when you route with nondefault wire widths.

For more information about the `prNonDefaultRules` subclass of the technology file, refer to "prNonDefaultRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`t_ruleName`	The name of the rule.
`t_propName`	The name of the property you want to set.
`g_value`	The value assigned to the property.

Return Values

`t`	The property was set.
`nil`	The technology file does not exist, the `prNonDefaultRules` subclass does not exist, or the rule is not defined in the `prNonDefaultRules` subclass.

techGetPrNonDefaultProps

```
techGetPrNonDefaultProps(
     d_techId
     t_ruleName
     )
     => l_nonDefaultProps/nil
```

Description

Returns the properties of the specified nondefault rule defined in the `prNonDefaultRules` subclass of the specified technology file. The `prNonDefaultRules` subclass is located in the Place and Route Rules class; it lists the rules that Preview Silicon Ensemble and Preview Gate Ensemble use when you route with nondefault wire widths.

For more information about the `prNonDefaultRules` subclass of the technology file, refer to "prNonDefaultRules" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
t_ruleName	The name of the rule.

Return Values

l_nonDefaultProps	A list of the properties defined for the via and the values you want to use for this via. The syntax is as follows: `( ( t_propName g_propValue ) ... )` *t_propName* is the name of the property you want to set. *t_propValue* is the value assigned to the property.
`nil`	The technology file does not exist, the `prNonDefaultRules` subclass does not exist, or the rule is not defined in the `prNonDefaultRules` subclass.

techSetPrRoutingPitch

```
techSetPrRoutingPitch(
     d_techFileID
     ltx_layer
     n_pitch
     )
     => t/nil
```

Description

Updates the routing pitch of the specified layer in the `prRoutingPitch` subclass of the specified technology file. The `prRoutingPitch` subclass is located in the Place and Route Rules class. Routing pitch is the minimum allowable spacing, center-to-center, between two regular geometries on different nets. If the specified layer is not listed in the `prRoutingPitch` subclass, this function appends the device and via type. If the `prRoutingPitch` subclass does not exist, this function creates it with the specified data.

For more information about the `prRoutingPitch` subclass of the technology file, refer to "prRoutingPitch" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`ltx_layer`	The layer to update. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
`n_pitch`	The routing pitch for the layer. Valid Values: any floating-point number, any integer

Return Values

`t`	The data was added to the `prRoutingPitch` subclass of the specified technology file.
`nil`	The technology file does not exist.

Example

```
techSetPrRoutingPitch( techFileID "metal1" 2.4)
=> t
```

Appends the specified routing layer data to the `prRoutingPitch` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techGetPrRoutingPitch

```
techGetPrRoutingPitch(
     d_techFileID
     ltx_layer
     )
     => n_pitch/nil
```

Description

Returns the routing pitch defined for the specified layer in the `prRoutingPitch` subclass of the specified technology file. The `prRoutingPitch` subclass is located in the Place and Route Rules class. Routing pitch is the minimum allowable spacing, center-to-center, between two regular geometries on different nets.

For more information about the `prRoutingPitch` subclass of the technology file, refer to "prRoutingPitch" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
ltx_layer	The layer for which to get the routing pitch. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

n_pitch	The routing pitch assigned to the layer.
`nil`	The technology file or the `prRoutingPitch` subclass does not exist.

Example

```
techGetPrRoutingPitch( techFileID "metal1" )
=> 2.4
```

Returns the routing pitch defined for the `metal1` layer in the `prRoutingPitch` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techSetPrRoutingOffset

```
techSetPrRoutingOffset(
     d_techFileID
     ltx_layer
     n_offset
     )
     => t/nil
```

Description

Updates the routing offset for the specified layer in the `prRoutingOffset` subclass of the specified technology file. The `prRoutingOffset` subclass is located in the Place and Route Rules class. Routing offset is the distance between the placement grid and the routing grid when there is a routing grid between two placement grids. If the layer is not listed in the `prRoutingOffset` subclass, this function appends the device and via type. If the `prRoutingOffset` subclass does not exist, this function creates it with the specified data.

For more information about the `prRoutingOffset` subclass of the technology file, refer to "prRoutingOffset" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`ltx_layer`	The layer to update. Valid Values: the layer name, the layer number, a list containing the layer name and purpose
`n_offset`	The routing offset for the layer. Valid Values: any floating-point number, any integer

Return Values

`t`	The data was added to the `prRoutingOffset` subclass in the specified technology file.
`nil`	The technology file does not exist or the specified layer does not exist.

Example

```
techSetPrRoutingOffset( techFileID "metal1" 0.0)
=> t
```

Appends the specified routing layer data to the `prRoutingOffset` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techGetPrRoutingOffset

```
techGetPrRoutingOffset(
     d_techFileID
     ltx_layer
     )
     => n_offset/nil
```

Description

Returns the routing offset defined for the specified layer in the `prRoutingOffset` subclass of the specified technology file. The `prRoutingOffset` subclass is located in the Place and Route Rules class. Routing offset is the distance between the placement grid and the routing grid when there is a routing grid between two placement grids.

For more information about the `prRoutingOffset` subclass of the technology file, refer to "prRoutingOffset" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.
ltx_layer	The layer for which you want the routing offset. Valid Values: the layer name, the layer number, a list containing the layer name and purpose

Return Values

n_offset	The routing offset assigned to the layer.
`nil`	The technology file does not exist, the `prRoutingOffset` subclass does not exist, or there is no offset defined for specified layer in the `prRoutingOffset` subclass.

Example

```
techGetPrRoutingOffset( techFileID "metal1" )
=> 0.0
```

Returns the routing offset defined for the `metal1` layer in the `prRoutingOffset` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techSetPrOverlapLayer

```
techSetPrOverlapLayer(
     d_techFileID
     ltx_overlapLayer
     )
     => t/nil
```

Description

Appends the specified layer to the `prOverlapLayer` subclass of the specified technology file (if it is not already in the `prOverlapLayer` list). The `prOverlapLayer` subclass is located in the Place and Route Rules class. You use the overlap layer to display the overlap boundary for Preview Silicon Ensemble cells. The overlap boundary indicates where cells cannot overlap. The purpose of the overlap layer is always `boundary`. If the `prOverlapLayer` subclass does not exist, this function creates it with the specified layer.

For more information about the `prOverlapLayer` subclass of the technology file, refer to "prOverlapLayer" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`ltx_overlapLayer`	The layer to set as a place-and-route overlap layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose, which is always `boundary`.

Return Values

`t`	The layer was set in the `prOverlapLayer` subclass in the specified technology file.
`nil`	The technology file does not exist.

Example

```
techSetPrOverlapLayer( techFileID "overlap" )
=> t
```

Appends the specified layer to the `prOverlapLayer` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techSetPrOverlapLayers

```
techSetPrOverlapLayers(
     d_techFileID
     l_overlapLayers
     )
     => t/nil
```

Description

Replaces the `prOverlapLayer` subclass of the specified technology file. The `prOverlapLayer` subclass is located in the Place and Route Rules class. You use an overlap layer to display the overlap boundary for Preview Silicon Ensemble cells. The overlap boundary indicates where cells cannot overlap. The purpose of an overlap layer is always `boundary`. If the `prOverlapLayer` subclass does not exist, this function creates it with the specified layer or layers.

For more information about the `prOverlapLayer` subclass of the technology file, refer to "prOverlapLayer" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

`d_techFileID`	The database identifier of the technology file.
`l_overlapLayers`	A list of the layers you want to define as overlap layers. The list has the following syntax: `list( ltx_overlapLayer ... )` *`ltx_overlapLayer`* is a layer to specify as an overlap layer. Valid Values: the layer name, the layer number, a list containing the layer name and purpose, which is always `boundary`.

Return Values

`t`	The layer was set in the `prOverlapLayer` subclass in the specified technology file.
`nil`	The technology file does not exist.

Example

```
techSetPrOverlapLayer( techFileID "overlap" "overlap2")
```

Sets the specified layers in the `prOverlapLayer` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

techGetPrOverlapLayers

```
techGetPrOverlapLayers(
     d_techFileID
     )
     => l_layerPurposes/nil
```

Description

Returns the names of all overlap layers defined in the `prOverlapLayer` subclass of the specified technology file. The `prOverlapLayer` subclass is located in the Place and Route Rules class. You use the overlap layer to display the overlap boundary for Preview Silicon Ensemble cells. The overlap boundary indicates where cells cannot overlap. The purpose of the overlap layer is always `boundary`.

For more information about the `prOverlapLayer` subclass of the technology file, refer to "prOverlapLayer" in the *Technology File and Display Resource File ASCII Syntax Reference Manual*.

Arguments

d_techFileID	The database identifier of the technology file.

Return Values

l_layerPurposes	A list of layer-purpose pairs. The list has the following syntax: `( ltx_layerPurpose )` *ltx_layerPurpose* is the layer name, layer number, or a list containing the layer name and purpose
`nil`	The technology file or the `prOverlapLayer` subclass does not exist.

Example

```
techGetPrOverlapLayer( techFileID )
=> overlap
```

Returns the `overlap` layer defined in the `prOverlapLayer` subclass in the Place and Route Rules class of the technology file identified by `techFileID`.

7

Display Resource File SKILL Functions

The functions in this chapter are used to change the display resource values in virtual memory. To save the changes you make using these Cadence® SKILL language functions, use the Display Resource Editor.

drDeleteDisplay

```
drDeleteDisplay(
     t_displayName
     )
     => t/nil
```

Description

Deletes the specified display device. You can use the drLoadDrf SKILL function to load a file containing this function.

Arguments

t_displayName	The display device name. Valid Values: any display device name

Return Values

t	The display device was deleted.
nil	The specified display device does not exist.

Example

```
drDeleteDisplay("psb")
=> t
```

Deletes the display device psb.

drDeleteColor

```
drDeleteColor(
     tx_display
     t_colorName
     )
     => t/nil
```

Description

Deletes the definition of the specified color for the specified display device from virtual memory. **Note:** The program does not check to see if any other definitions use this display device.

Arguments

tx_display	The display device name or identifier.
t_colorName	The color name.

Return Values

t	The specified color was deleted.
nil	The color does not exist for the specified display device.

Example

```
drDeleteColor("psb" "purple")
=> t
```

Deletes the color purple for the psb display device from virtual memory.

```
drDeleteColor(27832 "purple")
=> t
```

Deletes the color purple for the display device with the identifier 27832 from virtual memory.

drDeleteLineStyle

```
drDeleteLineStyle(
     tx_display
     t_lineStyleName
     )
     => t/nil
```

Description

Deletes the specified line style from virtual memory. **Note:** The program does not check to see if any of the packet definitions use this line style.

Arguments

tx_display	The display device name or identifier.
t_lineStyleName	The line style name.

Return Values

`t`	The line style was deleted.
`nil`	The line style does not exist for the specified display device.

Example

```
drDeleteLineStyle("psb" "solid")
=> t
```

Deletes the `solid` line style for the `psb` display device from virtual memory.

```
drDeleteLineStyle(27832 "solid")
=> t
```

Deletes the `solid` line style for the display device with the identifier `27832` from virtual memory.

drDeletePacket

```
drDeletePacket(
     tx_display
     t_packetName
     )
     => t/nil
```

Description

Deletes the definition of the specified packet for the specified display device from virtual memory. **Note:** The program does not check to see if any layer definitions use this packet.

Arguments

tx_display	The display device name or identifier.
t_packetName	The packet name.

Return Values

t	The specifed packet was deleted.
nil	The packet does not exist for the specified display device.

Example

```
drDeletePacket("psb" "yellow")
=> t
```

Deletes the `yellow` packet for the `psb` display device from virtual memory.

```
drDeletePacket(27832 "yellow")
=> t
```

Deletes the `yellow` packet for the display devicewith the identifier `27832` from virtual memory.

drDeleteStipple

```
drDeleteStipple(
     tx_display
     t_stippleName
     )
     => t/nil
```

Description

Deletes the definition of the specified stipple for the specified display device from virtual memory. **Note:** The program does not check to see if any of the packet definitions use this stipple.

Arguments

tx_display	The display device name or identifier.
t_stippleName	The stipple name.

Return Values

t	The stipple was deleted.
nil	The stipple does not exist for the specified display device.

Example

```
drDeleteStipple("psb" "dots")
=> t
```

Deletes the `dots` stipple for the `psb` display device from virtual memory.

```
drDeleteStipple(27832 "dots")
=> t
```

Deletes the `dots` stipple for the display devicewith the identifier `27832` from virtual memory.

drDumpDrf

```
drDumpDrf(
     t_fileName
     [g_saveChange]
     )
     => t/nil
```

Description

Dumps all of the display resource data from virtual memory or only the changes made in virtual memory into a file.

Arguments

t_fileName	The name of the file in which to save the display resource data.
g_saveChange	Indicates whether you want to save only the changes made in virtual memory (`t`) or all of the display resource data from virtual memory (`nil`). Valid Values: `t`, `nil` Default: `nil`

Return Values

`t`	The dump was successful.
`nil`	The dump was not successful.

Example

```
drDumpDrf( "/usr1/smith/display.drf" )
=> t
```

Saves all display resource data in virtual memory to the file `display.drf` in the directory `/usr1/smith`.

```
drDumpDrf( "/usr1/smith/display.drf" t )
=> t
```

Saves display resource data changed in virtual memory during the design session to the file `display.drf` in the directory `/usr1/smith`.

drFindPacket

```
drFindPacket(
     tx_display
     t_packetName
     )
     => l_packetList/nil
```

Description

Reads virtual memory and returns a list of attributes of the specified packet for the specified display device.

Arguments

tx_display	The display device name or identifier.
t_packetName	The packet name.

Return Values

l_packetList	A list containing the name of the display device, packet name, stipple, line style, fill color, outline color of the packet for the specified display device.
nil	The packet does not exist for the specified display device.

Example

```
drFindPacket("psb" "redsolid_S")
=> ("psb" "redsolid_S" "solid" "solid" "red" "red")
```

Reads virtual memory and returns the packet definition of `redsolid_S` for the `psb` display device.

```
drFindPacket(27832 "redsolid_S")
=> ("psb" "redsolid_S" "solid" "solid" "red" "red")
```

Reads virtual memory and returns the packet definition of `redsolid_S` for the display device with the identifier `27832`.

drGetColor

```
drGetColor(
     tx_display
     tx_color
     )
     => l_colorList/nil
```

Description

Reads virtual memory and returns the display device; color name; and the red, green, blue, and blink values.

Arguments

tx_display	The display device name or identifier.
tx_color	The color name or index number.

Return Values

l_colorList	A list containing the display device; color name; and the red, green, blue, and blink values.
nil	The color does not exist for the specified display device.

Example

```
drGetColor("psb" "purple")
drGetColor("psb" 7)
drGetColor(27832 "purple")
drGetColor(27832 7)
```

Reads virtual memory and returns the color definition of `purple` for the `psb` display device.

drGetDisplay

```
drGetDisplay(
     t_displayName
     )
     => x_displayID/nil
```

Description

Reads virtual memory and returns the display device identifier of the specified display device name.

Arguments

t_displayName	The display device name.

Return Values

x_displayID	The display device identifier.
nil	The specified display device does not exist.

Example

```
deGetDisplay("psb")
=> 27832
```

Reads virtual memory and returns the psb identifier 27832.

drGetDisplayIdList

```
drGetDisplayIdList( )
     => l_displayIDList/nil
```

Description

Reads virtual memory and returns a complete list of display device identifiers.

Arguments

None.

Return Values

l_displayIDList	The list of display device identifiers.
nil	No display devices exist.

Example

```
deGetDisplayIdList( )
=> 27832
```

Reads virtual memory and returns the display device identifiers (in this case, there is only one).

drGetDisplayName

```
drGetDisplayName(
     x_displayID
     )
     => t_displayName/nil
```

Description

Reads virtual memory and returns the display device name of the specified display device identifier.

Arguments

x_displayID	The display device identifier.

Return Values

t_displayName	The display device name.
nil	The specified display device identifier does not exist.

Example

```
drGetDisplayName(27832)
=> psb
```

Reads virtual memory and returns the display name, psb, for the display device with the identifier 27832..

drGetDisplayNameList

```
drGetDisplayNameList( )
     => l_displayNameList/nil
```

Description

Reads virtual memory and returns a complete list of display device names.

Arguments

None.

Return Values

l_displayNameList	The list of display device names.
nil	No display devices exist.

Example

```
deGetDisplayNameList( )
=> psb
```

Reads virtual memory and returns the display device names (in this case, there is only one).

drGetLineStyle

```
drGetLineStyle(
     tx_display
     tx_lineStyle
     )
     => l_lineStyleList/nil
```

Description

Reads virtual memory and returns the display device name and the line style name, thickness, and pattern.

Arguments

tx_display	The display device name or identifier.
tx_lineStyle	The line style name or index number.

Return Values

l_lineStyleList	A list containing the display device name and the line style name, thickness, and pattern.
nil	The line style does not exist for the specified display device.

Example

```
drGetLineStyle("psb" "solid")
drGetLineStyle("psb" 2)
drGetLineStyle(27832 "solid")
drGetLineStyle(27832 2)
```

Reads virtual memory and returns the line style definition of `solid` for the `psb` display device.

drGetLineStyleIndexByName

```
drGetLineStyleIndexByName(
     tx_display
     t_LineStyleName
     )
     => x_LineStyleIndex/nil
```

Description

Reads virtual memory and returns the line style index number for the specified line style for the specified display device.

Arguments

tx_display	The display device name or identifier.
t_LineStyleName	The line style name.

Return Values

x_LineStyleIndex	The line style index number.
nil	The line style does not exist for the specified display device.

Example

```
drGetLineStyleIndexByName("psb" "solid")
drGetLineStyleIndexByName(27832 "solid")
```

Reads virtual memory and returns the line style index 2.

drGetPacket

```
drGetPacket(
     tx_display t_packetName
     )
     => l_packetDefinition/nil
```

Description

Reads virtual memory and returns the definition of the specified display packet for the specified display device.

Arguments

tx_display	The display device name or identifier.
t_packetName	The display packet name.

Return Values

l_packetDefinition	A list containing the display packet definition. The list has the following syntax: `(t_displayName t_packetName t_stippleName t_lineStyleName t_fillColor t_outlineColor t_fillStyle )` *t_displayName* is the name of the display device. *t_packetName* is the name of the display packet. *t_stippleName* is the name of the stipple pattern. *t_lineStyleName* is the name of the line style. *t_fillColor* is the name of the fill color. *t_outlineColor* is the name of the outline color. *t_fillStyle* is the name of the fill style.

`nil`	The specified display device has no display packets associated with it or the specified display device or display packet does not exist.

Example

```
drGetPacket("display" "hardFence")
("display" "hardFence" "blank" "solid" "red" "red" "outline")
```

Reads virtual memory and returns the definition of the display packet `hardFence` for the display device `display`; the stipple pattern is `blank`, the line style is `solid`, the fill color is `red`, the outline color is `red`, and the fill style is `outline`.

drGetPacketList

```
drGetPacketList(
     tx_display
     )
     => l_packetName/nil
```

Description

Reads virtual memory and returns a list of the names of all of the display packets defined for the specified display device.

Arguments

tx_display	The display device name or identifier.

Return Values

l_packetName	A list containing the names of all of the display packets defined for the specified display device.
nil	The specified display device has no display packets associated with it or does not exist.

Example

```
drGetPacketList("psb")
drGetPacketList(27832)
```

Reads virtual memory and returns the list of display packets assigned to the psb display device.

drGetPacketAlias

```
drGetPacketAlias(
     tx_displayName
     t_srcPacketName
     )
     => l_packetAliasList/nil
```

Description

Reads virtual memory and returns a list of packets that are aliased to the specified packet.

Arguments

tx_display	The display device name or identifier.
t_srcPacketName	The packet name.

Return Values

l_packetAliasList	A list containing the name of the display device, specified packet, and packet aliases.
nil	The packet does not exist for the specified display device.

Example

```
drGetPacketAlias("psb" "blackChecker_S")
drGetPacketAlias(27832 "blackChecker_S")
```

Reads virtual memory and returns the packets aliased to the `blackChecker_S` packet for the `psb` display device.

drGetPacketFillStyle

```
drGetPacketFillStyle(
     tx_display
     t_packetName
     )
     => x_fillStyle/nil
```

Description

Reads virtual memory and returns the fill style number of the specified packet for the specified display device.

Arguments

tx_display	The display device name or identifier.
t_packetName	The packet name.

Return Values

x_fillStyle	The fill style number.
nil	The packet does not exist for the specified display device.

Example

```
drGetPacketFillStyle("psb" "greenbluedots_L")
drGetPacketFillStyle(27832 "greenbluedots_L")
```

Reads virtual memory and returns the fill style number of the packet named greenbluedots_L for the psb display device. The fill style numbers have the following meanings:

Number	Meaning	Number	Meaning
0	Unknown	3	Filled in with an X
1	Not filled in, only outlined	4	Filled in with a pattern
2	Filled in with color	5	Filled in with a pattern and outlined

drGetStipple

```
drGetStipple(
     tx_display
     tx_stipple
     )
     => l_stippleList/nil
```

Description

Reads virtual memory and returns the display device name and the stipple name, width, height, and pattern.

Arguments

tx_display	The display device name or identifier.
tx_stipple	The stipple name or index number.

Return Values

l_stippleList	The display device name and the stipple name, width, height, and pattern.
nil	The stipple does not exist for the specified display device.

Example

```
drGetStipple("psb" "dots")
drGetStipple("psb" 3)
drGetStipple(27832 "dots")
drGetStipple(27832 3)
```

Reads virtual memory and returns the stipple definition of `dots` for the `psb` display device.

drGetStippleIndexByName

```
drGetStippleIndexByName(
     tx_display
     t_stippleName
     )
     => x_stippleIndex/nil
```

Description

Reads virtual memory and returns the stipple index number.

Arguments

tx_display	The display device name or identifier.
t_stippleName	The stipple name.

Return Values

x_stippleIndex	The stipple index number.
nil	The stipple does not exist for the specified display device.

Example

```
drGetStippleIndexByName("psb" "dots")
drGetStippleIndexByName(27832 "dots")
```

Reads virtual memory and returns the stipple index 3.

drLoadDrf

```
drLoadDrf(
     t_filename [g_askToSave]
     )
     => t/nil
```

Description

Loads the display resource file (usually called `display.drf`) from any location.

Arguments

t_filename	The path and name of the display resource file (usually called `display.drf`).
g_askToSave	Indicates whether you want a dialog box displayed asking whether you want to save your changes when you quit the Display Resource Editor. Valid Values: `t`, `nil` Default: `t`

Return Values

`t`	The specified file was loaded into virtual memory.
`nil`	The file does not exist.

Example

```
drLoadDrf( "~/display.drf" )
=> t
```

Loads the `display.drf` file from your home directory.

drMakeColor

```
drMakeColor(
     tx_display
     t_colorName
     l_colorValues
     x_index
     )
     => t/nil
```

****Description

Loads the display resource file (usually called `display.drf`) from any location.

Arguments

t_filename	The path and name of the display resource file (usually called `display.drf`).
g_askToSave	Indicates whether you want a dialog box displayed asking whether you want to save your changes when you quit the Display Resource Editor. Valid Values: `t`, `nil` Default: `t`

Return Values

`t`	The specified file was loaded into virtual memory.
`nil`	The file does not exist.

Example

```
drLoadDrf( "~/display.drf" )
=> t
```

Loads the `display.drf` file from your home directory.

drSetPacket

```
drSetPacket(
     tx_display
     t_packetName
     t_stippleName
     t_lineStyleName
     t_fillColorName
     t_outlineColorName
     )
     => t/nil
```

Description

Updates the value of the specified packet for the specified display device in virtual memory.

Arguments

tx_display	The display device name or identifier.
t_packetName	The packet name.
t_stippleName	The stipple name.
t_lineStyleName	The line style name.
t_fillColorName	The fill color name.
t_outlineColorName	The outline color name.

Return Values

`t`	The packet values were updated.
`nil`	The packet does not exist for the specified display device.

Example

```
drSetPacket("psb" "bluethin_L" "blank" "thin" "blue" "tan")
=> t
```

Sets the values for the `bluethin_L` packet as `blank` stipple, `thin` line, `blue` fill and `tan` outline for the `psb` display device in virtual memory.

```
drSetPacket(27832 "bluethin_L" "blank" "thin" "blue" "tan")
=> t
```

Sets the values for the `bluethin_L` packet as `blank` stipple, `thin` line, `blue` fill and `tan` outline for the display device with the identifier `27832` in virtual memory.