cadence®

Virtuoso Technology Data SKILL Reference

Product Version IC23.1
June 2023

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

Contents

1

Administrative Functions

The administrative SKILL functions 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 Virtuoso® Design Environment library
- 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
- Determine the units set for each cellview type in a technology database

The functions are categorized as:

- File Functions

  These functions let you open, close, and manage data 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 as follows:

  - Read — loads the contents of the technology database into virtual memory and does not allow any edits to it
  - Append — loads the contents of the technology database into virtual memory and does allow edits

- Write — deletes the contents of the technology database on disk and loads the empty database into virtual memory. Write mode does allow edits. Use this mode with extreme caution.

The file functions are listed below:

- `techOpenTechFile`
- `techOpenDefaultTechFile`
- `techReopenTechFile`
- `techCopyTechFile`
- `techDeleteTechFile`
- `techSaveTechFile`
- `techCloseTechFile`
- `techPurgeTechFile`
- `techRefreshTechFile`
- `techTruncateTechFile`
- `techGetDefaultTechName`
- `techGetTechFile`
- `techGetTechFileDdId`
- `techGetOpenTechFiles`
- `techVerifyTechFileId`
- `techSetEvaluate`
- `techSetTimeStamp`
- `techGetTimeStamp`

■ Attach Functions

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

The technology database binding is made up of two properties, `techLib` and `techFile`. You must set both properties for every DFII library. The properties are defined as follows:

`techLib`	The name of the DFII technology database.
`techFile`	The name of the binary technology database.

The attach functions are listed below:

- `techBindTechFile`
- `techGetTechFileName`
- `techSetTechLibName`
- `techGetTechLibName`
- `techDeleteTechLibName`
- `techUnattachTechFile`

■ Dump, Load, and Trigger Functions

These functions 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, pre-event and post-event. Pre-event triggers notify you before the event occurs, post-event triggers notify you afterward.

The dump, load, and trigger functions are listed below:

- `tcDumpTechFile`
- `tcLoadTechFile`
- `tcRegPostAttachTrigger`
- `tcUnregPostAttachTrigger`
- `tcRegPreDumpTrigger`
- `tcUnregPreDumpTrigger`
- `tcRegPostDumpTrigger`
- `tcUnregPostDumpTrigger`

- `tcRegPreLoadTrigger`
- `tcUnregPreLoadTrigger`
- `tcRegPostLoadTrigger`
- `tcUnregPostLoadTrigger`
- `tcRegPostSetRefTrigger`
- `tcUnregPostSetRefTrigger`

- Floating-Point Precision Function

  This function 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` is a floating-point function.:

- ICC Information Function

  This function creates a constraint group and rules file for the Virtuoso chip assembly router.

  `techMakeVirtuosoIccInfo` is an ICC information function.

tcDumpTechFile

```
tcDumpTechFile(
     d_techID
     t_dumpFile
     [ l_sectionList t_mode
     g_dumpEmptySectionHdrs ]
     )
     => t / nil
```

Description

Dumps the technology data in the specified binary technology database to the specified ASCII file.

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

Arguments

d_techID	The identifier of the technology database to dump.
t_dumpFile	Name of the ASCII file to create.
l_sectionList	List of the technology file sections to dump. The list has the following syntax: `list( t_sectionName … )` where, *t_className* is the name of the main technology file section.
t_mode	Mode in which to dump the specified sections: append to the specified dump file or write over the specified dump file if it already exists. Valid values: `a`, `w`
g_dumpEmptySectionHdrs	Dumps all section headers for sections selected in the section list. If the section list is `nil`, then it will dump all the sections.

Value Returned

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

Example

```
tcDumpTechFile(tfID "dumpFile")
```

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

```
tcDumpTechFile(tfID "dumpFile" list("controls" "constraintGroups" "viaDefs") "w")
```

Dumps the contents of the `controls`, `constraintGroups`, and `viaDefs` sections of the technology database identified by `tfID` to a text file called `dumpFile` in write mode (if the file `dumpFile` already exists, this function overwrites it).

tcLoadTechFile

```
tcLoadTechFile(
     d_techID
     t_sourceFile
     [ t_mode [ l_sections ] ]
     )
     => t / nil
```

Description

Compiles the ASCII source file, opens the technology database in `a` (append) mode, and updates it with the compiled data. The default action is to merge the newly compiled data with the technology database 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 sections to compile, this function compiles only the specified sections and preserves the data in the other sections.

If you opened the technology database in read mode, this function upgrades the mode to append or write. If you opened the technology database 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_techID`	Database identifier of the technology database.
`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_sections`	List of sections to compile. Valid values: ‘`controls`, ‘`layerDefinitions`, ‘`layerRules`, ‘`constraintGroups`, ‘`siteDefs`, ‘`viaDefs`, ‘`viaSpecs`, ‘`devices`, ‘`leRules`

Value Returned

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

Example

```
tcLoadTechFile(techGetTechFile( techID ) "my_source")
```

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

tcRegPostAttachTrigger

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

Description

Registers a trigger function the system calls after attaching a design library to a technology library. You can use this function to keep track of the technology library of a design library is attached to.

Arguments

s_function	A SKILL procedure with two arguments: the name of the design library and the name of the technology database.
x_priority	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). Valid values: Any integer

Value Returned

`t`	The trigger function is registered.
`nil`	The trigger function is registered.

Example

```
procedure(myPostAttachTrig(libName techLibName)
    println("myPostAttachTrig")
    printf("Attach design library '%s' to technology library '%s'.\n" libName
techLibName)
)
tcRegPostAttachTrigger('myPostAttachTrig)
```

Registers a trigger function, `'myPostAttachTrig`.

tcRegPostDumpTrigger

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

Description

Registers a trigger function the system calls after it dumps a technology database. The *s_function* argument is a SKILL procedure with three arguments: the database identifier of the technology database, 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 as follows:
	`function( d_techID p_port t_dumpName )`
	`=> t / nil`
	where,
	■ *d_techID* is the database identifier of the technology database.
	■ *p_port* is the port for the dump file. (You can use the `infile` function to get port data for a file.)
	■ *t_dumpName* is the name of the dump file.

x_priority	Priority of this trigger function. Valid values: Any integer

Value Returned

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

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 database 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 as follows:

```
function( d_techId t_techFileName )
=> t / nil
```

where,

- *d_techId* is the database identifier of the technology database.
- *t_techFileName* is the name of the ASCII technology file.

Returns `t` if the database identifier is loaded successfully; returns `nil` if the database identifier or ASCII technology file does not exist.

x_priority	Priority of this trigger function, with 0 having the highest priority. If you do not specify a priority argument, by default it takes the lowest priority. It means, this function is executed after any other trigger function that specifies a priority. Valid values: Any integer

Value Returned

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

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

tcRegPostSetRefTrigger

```
tcRegPostSetRefTrigger(
     s_trigFunc
     [ x_priority ]
     )
     => t / nil
```

Registers a trigger function that the system calls after setting the reference for the specified technology database. You can use this function to keep track of the technology database reference when a technology database is updated.

Arguments

s_trigFunc	Name of the trigger.
x_priority	Priority of the trigger. 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).

Value Returned

`t`	The trigger function is registered successfully.
`nil`	The trigger function is not registered. The SKILL procedure either does not exist or is incomplete.

Example

```
procedure(myPostSetRefTrig(libName)
    println("myPostSetRefTrig")
    tf =  techGetTechFile(ddGetObj(libName))
    printf("Set Ref '%L' to techLib '%s'.\n" tf~>refLibNames libName)
)
tcRegPostSetRefTrigger('myPostSetRefTrig)
=> t
```

Registers the `myPostSetRefTrig` trigger.

tcRegPreDumpTrigger

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

Description

Registers a trigger function the system calls before dumping a technology database. The *s_function* argument is a SKILL procedure with three arguments: the database identifier of the technology database, 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 as follows:

```
function( d_techID p_port t_dumpName )
=> t / nil
```

where,

- *d_techID* is the database identifier of the technology database.
- *p_port* is the port for the dump file. (You can use the infile function to get port data for a file.)
- *t_dumpName* is the name of the dump file.

x_priority

Priority of this trigger function.

Valid values: Any integer

Value Returned

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

Example

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

tcRegPreDumpTrigger('MYPreDumpTrigger)
```

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

tcRegPreLoadTrigger

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

Description

Registers a trigger function the system calls before loading a technology database. The *s_function* argument is a SKILL procedure with two arguments: the database identifier of the technology database 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 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 as follows:

```
function( d_techID t_techFileName )
=> t / nil
```

where,

- *d_techID* is the database identifier of the technology database.
- *t_techFileName* is the name of the ASCII technology file.

x_priority

Priority of this trigger function.

Valid values: Any integer

Value Returned

`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.

tcUnregPostAttachTrigger

```
tcUnregPostAttachTrigger(
     s_trigFun
     )
     => t / nil
```

Description

Unregisters post attach trigger so the system does not call the function after attaching a design library to a technology library.

Arguments

s_trigFun	Name of the trigger.

Value Returned

`t`	The trigger function is unregistered successfully.
`nil`	An error occurred during execution.

Example

```
tcUnregPostAttachTrigger('myPostAttachTrig)
```

Unregisters a trigger function, `'myPostAttachTrig`.

tcUnregPostDumpTrigger

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

Description

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

Arguments

s_function	Symbol indicating the name of the registered trigger function.

Value Returned

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

Example

```
tcUnregPostDumpEchFileTrigger('MYPostLoadTriggerFunc)
```

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.

Value Returned

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

Example

```
tcUnregPostLoadTrigger('MYPostLoadTriggerFunc)
```

Unregisters `MYPostLoadTriggerFunc` trigger function.

tcUnregPostSetRefTrigger

```
tcUnregPostSetRefTrigger(
     s_trigFunc
     )
     => t / nil
```

Unregisters a specified trigger function so that the system does not call it after setting the reference for the specified technology database.

Arguments

s_trigFunc	Name of the technology database.

Value Returned

t	The trigger function is unregistered successfully.
nil	An error occurred during execution.

Example

```
tcUnregPostSetRefTrigger('CCSpostSetRefTrig)
=> t
```

Unregisters the CCSpostSetRefTrig trigger.

tcUnregPreDumpTrigger

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

Description

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

Arguments

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

Value Returned

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

Example

```
tcUnregPreDumpTrigger('MYPreDumpTriggerFunc)
```

Unregisters `MYPreDumpTriggerFunc` trigger function.

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.

Value Returned

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

Example

```
tcUnregPreLoadTrigger('MYPreLoadTriggerFunc)
```

Unregisters `MYPreLoadTriggerFunc` trigger function.

techBindTechFile

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

Description

Attaches the specified DFII library to the specified technology database 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`, see *Virtuoso Studio Design Environment SKILL Reference*.

Arguments

g_ID	The design data identifier for the library to attach.
t_techLibName	The design data identifier for the DFII library. Default: `cdsDefTechLib`
t_binaryName	The name of the binary technology database. You must specify a technology library name for *t_techLibName* before you can use this argument. Default: `tech.db`
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_binaryName* before you can use this argument. Valid values: `t`, `nil` Default: `nil`

Value Returned

`t`	The `techFile` and `techLib` properties are set to the specified technology database.
`nil`	The DFII library or technology database does not exist.

Example

```
techBindTechFile(ddGetObj("myLib") "newTechLib" "tech.db" t)
=> t
```

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

techCloseTechFile

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

Description

Changes the status of the technology database to closed and decrements the close count. The technology database 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 database. 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 database from virtual memory.

Arguments

d_techID	The identifier of the technology database.

Value Returned

t	The technology database was closed.
nil	The technology database does not exist.

Example

```
techCloseTechFile(tfID)
=> t
```

Closes the technology database identified by `tfID`.

techCopyTechFile

```
techCopyTechFile(
     d_techID
     t_newTechDBName
     t_path
     [ g_deleteOriginal ]
     )
     => d_newtechID / nil
```

Description

Copies a technology database 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 as it will copy only the technology data.

Arguments

d_techID	The identifier of the technology database.
t_newTechDBName	The name of the new technology database to create.
t_path	The UNIX path of the directory where you want the new technology database 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 database attached to the new technology database and the original technology data removed from the design library. Valid values: `t`, `nil` Default: `nil`

Value Returned

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

Example

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

Copies the technology database identified by `tfID` to a new database called `newTechLib` and reattaches the contents of the original database to the new technology database.

techDeleteTechFile

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

Description

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

Arguments

d_techID	The identifier of the technology database.

Value Returned

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

Example

```
techDeleteTechFile( tfID )
=> t
```

Deletes the technology database identified by `tfID`.

techDeleteTechLibName

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

Description

Deletes the techLib property for the specified DFII library. This property is one of two that attach a library to a technology database.

DFII applications cannot display a cellview without a technology database. If you delete part of the technology database binding, DFII uses the default technology database your_install_dir/tools/dfII/etc/cdsDefTechLib/tech.db.

Arguments

g_ID	The design data identifier for the library. To get the design data identifier, use ddGetObj. For more information about design data identifiers and ddGetObj, see *Virtuoso Studio Design Environment SKILL Reference*.

Value Returned

t	The techLib property is deleted.
nil	The library 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").

techGetDefaultTechName

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

Description

Returns the default name of the binary technology database.

Arguments

None

Value Returned

t_binaryName	The default name of the binary technology database (always `tech.db`).
`nil`	The function failed.

Example

```
techGetDefaultTechName( )
=> "tech.db"
```

Returns the default binary technology database name, `tech.db`.

techGetOpenTechFiles

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

Description

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

Arguments

None

Value Returned

l_techIDs	A list of database identifiers for the open technology databases.
nil	There are no technology databases open.

Example

```
techGetOpenTechFiles()
=> (db:0x025c001a db:0x025c001b db:0x025c001d db:0x025c0030)
```

Returns the database identifiers of all open technology databases.

techGetTechFile

```
techGetTechFile(
     { g_libID }
     )
     => d_techID / nil
```

Description

Returns the identifier of the technology database attached to the specified DFII library.

Arguments

g_libID	The design data identifier for the database. To get the design data identifier, use the `ddGetObj` function. For more information, see *Virtuoso Studio Design Environment SKILL Reference*.

Value Returned

d_techID	The database identifier of the technology database.
`nil`	The library does not exist.

Example

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

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

techGetTechFileDdId

```
techGetTechFileDdId(
     d_techID
     )
     => g_ddtechID / nil
```

Description

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

For more information about design data identifiers and ddGetObj, see *Virtuoso Studio Design Environment SKILL Reference*.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

g_ddtechId	The design data identifier of the technology database.
nil	The technology database does not exist.

Example

```
ddtechFileId = techGetTechFileDdId ( tfID )
```

Returns the design data identifier of the technology database identified by tfID.

techGetTechFileName

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

Description

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

Arguments

g_ID	The design data identifier for the library. To get the design data identifier, use `ddGetObj`. For more information, see *Virtuoso Studio Design Environment SKILL Reference*.

Value Returned

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

Example

```
techGetTechFileName(ddGetObj("libName"))
=> tech.db
```

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

techGetTechLibName

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

Description

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

Arguments

g_ID	The design data identifier for the library. To get the design data identifier, use `ddGetObj`. For more information about design data identifiers and `ddGetObj`, see *Virtuoso Studio Design Environment SKILL Reference*.

Value Returned

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

Example

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

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

techGetTimeStamp

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

Description

Returns the last save time of the technology database loaded. The time stamp is updated every time you save the technology database. The time stamp is an integer representing the number of seconds elapsed since 00:00:00 GMT, January 1, 1970.

Arguments

d_techID	The identifier of the technology database.

Value Returned

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

Example

```
techGetTimeStamp( tfID )
=> 10281600
```

Returns the time elapsed since the last update of the technology database identified by `tfID`. In this example, the time is 10281600/60*60*24 = 119 days.

techMakeVirtuosoIccInfo

```
techMakeVirtuosoIccInfo(
     d_techID
     [ g_iccRuleFileName ]
     [ g_constraintGroupName ]
     [ b_replaceConstraintGroup ]
     [ b_removeParams ]
     )
     => t / nil
```

Description

Creates a constraint group from Virtuoso chip assembly router rules. Also creates a separate abstract rules file when `iccConductors` and/or `iccKeepouts` are specified. The router rules can be either in a separate ASCII icc rules file or in the technology file `techParams` section.

Arguments

d_techID	The database identifier of the technology database in which to create the constraint group
g_iccRuleFileName	The name of the Virtuoso chip assembly router rules file. Valid values: A Virtuoso chip assembly router rules filename or `nil` to take data from technology file `techParams` section. Default: `nil`
g_constraintGroupName	The name of the constraint group to create. Valid values: A valid constraint group name or `nil`. Default: `virtuosoDefaultSetup`
b_replaceConstraintGroup	If the named constraint group already exists, indicates whether to replace the named constraint group or append to it. Valid values: `t` (replaces) and `nil` (appends) Default: `nil`
b_removeParams	If parameters are obtained from the technology file, instead of a chip assembly router rules file, indicates whether to remove the parameters from the `techParams` section after mapping them to a constraint group. Valid values: `t` (remove) and `nil` (retain) Default: `nil`

Value Returned

`t`	The constraint group and separate abstract file, if required, were successfully created.
`nil`	The technology database does not exist, the router rules file does not exist, or there is no data from which to create the constraint group.

Example

```
techMakeVirtuosoIccInfo(tfID "iccRules")
=> t
```

Creates the constraint group `virtuosoDefaultSetup` in the technology database identified by `tfID` from the router rules file `iccRules`.

techOpenDefaultTechFile

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

Description

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

your_install_dir/tools/dfII/etc/cdsDefTechLib/tech.db

Arguments

None

Value Returned

d_techID	The identifier of the technology database loaded into virtual memory.
nil	The technology database does not exist or is not at the expected location.

Example

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

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

techOpenTechFile

```
techOpenTechFile(
     t_libName
     t_binaryName
     [ t_mode ]
     )
     => d_techID / nil
```

Description

Loads a binary technology database in virtual memory with the specified mode (`r`, `w`, and `a`) and returns the database identifier of that database.

- If you specify `r` mode, the file can only be read.
- If you specify `w` mode, the contents of the database are deleted or truncated, and an empty technology database is loaded into virtual memory.
- If you specify `a` mode, the technology database is loaded in append mode, which means that you can edit the contents of the database.

The `libName` argument specifies the library that the technology database will be loaded from. If the specified library contains the technology database, that database will be opened using the requested mode. If the library is a design library that is attached to another library for technology data, the technology database from the attached technology library will be opened. If the library is a design library that is attached to the default Cadence technology database, then if the mode requested is '`r`', the default Cadence technology database will be opened. If the library is a design library that is attached to the default Cadence technology database, if the mode is '`a`' or '`w`', an empty technology database will be created in the library.

The `libName` of the `dbId` for the technology database can be queried after the function is complete to determine if the technology database loaded is defined in the requested library or is from an attached technology library.

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

Arguments

t_libName	Name of the library from which the technology database is opened.
t_binaryName	The name of the binary technology library to open. This argument is no longer used, but needs to be specified to pass the syntax check. Valid values: Any string
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) Default: `r` **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 binary technology database is loaded in append mode, which means that you can edit the contents of the file.

Value Returned

d_techID	Database identifier of the technology database loaded into virtual memory.
`nil`	The technology library or file does not exist.

Example

```
techID = techOpenTechFile("cellTechLib" "tech" "a")
=> db:25675212
```

Opens the technology database from the library named "`cellTechLib`", and loads it into virtual memory in append mode. Sets the variable `techID` to that database.

techPurgeTechFile

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

Description

Deletes the database 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 database. 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 database to free the memory. This function lets you manually purge the technology database.

Arguments

d_techID	The identifier of the technology database.

Value Returned

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

Example

```
techPurgeTechFile(tfID)
=> t
```

Frees the memory allocated to the technology database identified by `tfID`.

techRefreshTechFile

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

Description

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

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

Arguments

d_techID	The identifier of the technology database.

Value Returned

`t`	The technology database was refreshed.
`nil`	The technology database does not exist.

Example

```
techRefreshTechFile(tfID)
=> t
```

Refreshes the technology database identified by `tfID`.

techReopenTechFile

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

Description

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

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 database contents are not refreshed and you are not prompted to save your changes when you exit.

Arguments

d_techID	The identifier of the technology database.
t_mode	The mode in which to reopen the database. Valid values: `a` (append or edit mode), `r` (read only), `w` (delete contents and load empty file)

Value Returned

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

Example

```
techReopenTechFile(techID "a")
=> t
```

Reopens the technology database identified by `techID` in `a` (append) mode so you can edit it.

techSaveTechFile

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

Description

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

Arguments

d_techID	The identifier of the technology database.

Value Returned

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

Example

```
techSaveTechFile( tfID )
=> t
```

Saves the technology database identified by `tfID`.

techSetEvaluate

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

Description

Sets an internal flag that indicates whether the `tcDumpTechFile` and `techGet` functions evaluate expressions or read expressions as strings. The system automatically sets this internal flag to `nil` when 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, see Controls Functions.

Arguments

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

Value Returned

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

Example

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

Controls section of the loaded technology file

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

Entered in the CIW

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

Output in the CIW

```
3.8
```

Entered in the CIW

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

Output in the CIW

```
(techParam("WIDTH") + 3.0)
```

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.

This function treats a real number as a float (double) number and performs the following checks.

- If absolute value of number less than equal to `1/10^n` (precision), then the same number is returned.
- If absolute value of number greater than precision, then it is rounded off to the precision digit.

Arguments

x_digits	Number of digits following the decimal point in floating-point numbers. Valid values: `1` through `8` **Note:** If a value greater than `8` is assigned, *x_digit* is set to `8`. It is also ensured that no overflow occurs when the value gets close to `8`.

Value Returned

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

Example

```
techSetPrecision(6)
```

Sets the precision for floating-point numbers to `6`.

techSetTechLibName

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

Description

Updates the techLib property of the specified library. This property is one of two that attach a library to a technology database.

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. To get the design data identifier, use ddGetObj. For more information, see *Virtuoso Studio Design Environment SKILL Reference*.
t_libName	The name of the DFII library containing the technology database you want to use.

Value Returned

t	The techLib property is updated.
nil	The library does not exist.

Example

```
techSetTechLibName(libId "techLibName")
=> t
```

Updates the techLib property for the library identified bylibID.

techSetTimeStamp

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

Description

Updates the internal time stamp of the technology database 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_techID	The identifier of the technology database.

Value Returned

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

Example

```
techSetTimeStamp(tfID)
=> t
```

Updates the time stamp for the technology database identified by `tfID` to the current time.

techTruncateTechFile

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

Description

Deletes the contents of the technology database stored on disk.

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

Arguments

d_techID	The identifier of the technology database.

Value Returned

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

Example

```
techTruncateTechFile(tfID)
=> t
```

Truncates the technology database identified by `tfID`.

techUnattachTechFile

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

Description

Unattaches a technology database from a design library. This function unattach the technology library from the design library without remaster the devices in the design library. However, the device master cellviews remain unchanged.

If you unattach the technology database from a design library, without attaching another technology database to the design library, DFII uses the default technology database your_install_dir/tools/dfII/etc/cdsDefTechLib/tech.db.

Arguments

d_object	The design data identifier for the library for which to unattach its attached technology library. To get the design data identifier, use `ddGetObj`. For more information, see *Virtuoso Studio Design Environment SKILL Reference*.

Value Returned

`t`	The technology library is unattached from the design object.
`nil`	The librarydoes not exist.

Example

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

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

techVerifyTechFileId

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

Description

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

Arguments

d_ID	The database identifier you want to check.

Value Returned

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

Example

```
techVerifyTechFileId( tfID )
=> t
```

Indicates that the variable `tfID` contains the database identifier of a technology database.

2

Controls Functions

The control functions let you set global parameters and permissions that affect all or selected sections of the technology file.

The `techParams` control lets you set parameters that you can use in the other technology file sections. The function `techSetEvaluate` lets you specify whether or not to evaluate parameters when you use `techGet` functions or dump the technology file. By default, no evaluation is done.

The `techPermissions` control lets you set read-only and read/write permissions on sections of your technology file for different system users. Users can then access technology file sections only as their assigned permissions allow.

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 section 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)
```

techGetFabricType

```
techGetFabricType(
     d_techfileID
     )
     => t_fabricType / nil
```

Description

Returns the fabric type of a technology database.

Arguments

d_techfileID	The database identifier of the technology database.

Value Returned

t_fabricType	The fabric type of the technology database.
nil	The technology database does not exist, or does not have a fabric type specified.

Example

```
techGetFabricType(tech)
=> "ic"
```

Returns `ic` as the fabric type for the technology database.

techGetMfgGridResolution

```
techGetMfgGridResolution(
     d_techID
     )
     => g_resolution / nil
```

Description

Returns the value of the manufacturing grid resolution defined in the specified technology database. ASCII technology file location: `mfgGridResolution` subsection in the `controls` section. When specified, it establishes that grid snapping must be a multiple of the value of *`g_resolution`*.

For more information about the `mfgGridResolution` subsection of the technology file, see mfgGridResolution.

Arguments

`d_techID`	The identifier of the technology database.

Value Returned

`g_resolution`	The number of user units to use as the basis for the grid, as specified in the `mfgGridResolution` subsection of the specified technology file.
`nil`	The technology file does not exist or no manufacturing grid resolution is specified.

Example

```
techGetMfgGridResolution(tfID)
=> 0.005
```

Returns the value `0.005`, which is the manufacturing grid resolution specified in the technology file with the database identifier stored in `tfID`.

techGetParam

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

Description

Returns the value of the named control parameter in the specified technology database.

Arguments

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

Value Returned

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

Example

```
tfID = techGetTechFile(ddGetObj("testLib"))
db:0x0180200d
techGetParam(tfID "lambda")
=> 0.6
```

Returns the value defined for the control parameter `lambda` in the technology file identified by `tfID` (`testLib`).

techGetParams

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

Description

Returns the list of control parameters that are set in the specified technology database.

Arguments

d_techFileID	The database identifier of the technology file.

Value Returned

l_params	List of control parameters and their values. The list has the following syntax: `( ( t_paramName g_paramValue ) … )` where, ■ *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

```
tfID = techGetTechFile(ddGetObj("testLib"))
db:0x0180200d
techGetParams(tfID)
=> (("lambda" 0.6)
        ("theta" 2.5)
   )
```

Returns the parameters defined in the technology file identified by `tfID` (`testLib`).

techGetPermission

```
techGetPermission(
     d_techFileID
     t_sectionName
     )
     => l_permissions / nil
```

Description

Returns the permissions explicitly applied to the specified section in the specified technology database.

Arguments

d_techFileId	The database identifier of the technology file.
t_sectionName	The name of the technology file section.

Value Returned

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

Example

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

Gets and displays the permissions assigned to the `layerDefinitions` section 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`).

```
techGetPermission(tfID "constraintGroups")
=> (("fred" "jim")
      ("mary" "bob")
)
```

Lists the permissions explicitly assigned to the `contraintGroups` section of the technology file identified by `tfID`.

techGetPermissions

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

Description

Returns the permissions explicitly applied to the specified technology database.

Arguments

d_techFileId	The database identifier of the technology file.

Value Returned

l_permissions	A list of the permissions set on the technology file.
	
	The software returns the permissions for any section with read-only or read/write permissions explicitly set. The following defines the format in which the software returns the information:
	
	`((sectionName1`
	`  (list_of_read-only_names) | nil`
	`  (list_of_read/write_names) | nil`
	`)`
	`(sectionName2`
	`  (list_of_read-only_names) | nil`
	`  (list_of_read/write_names) | nil`
	`)`
	`)`
	
	Note: The software may return the names of both users with read-only permissions and 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.

`nil`	The technology file does not exist, or no permissions are explicitly set on it.

Example

```
techGetPermissions( tfID )
=> (
    ("layerDefinitions"
           ("joe")
              nil
     )
    ("devices"
           ("joe")
           ("mary")
     )
    ("constraintGroups"
           ("fred" "jim")
           ("mary" "bob")
     )
  )
```

Displays the permissions assigned to the technology file with the database identifier stored in `tfID`. Three sections in this technology file have permissions explicitly assigned. For the `layerDefinitions` section,

- 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` section,

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

For the constraints groups,

- Users `fred` and `jim` have read-only permission
- Users `mary` and `bob` are specifically assigned read/write permission
- All other users have read/write permission by default

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

techGetProcessNode

```
techGetProcessNode(
     d_techFileID
     )
     => f_processNode / nil
```

Description

Returns the local processNode value of the specified technology database. The function does not consider incremental technology databases.

Arguments

d_techFileID	The database identifier of the technology file.

Value Returned

f_processNode	The processNode value (in user units).
nil	The technology file does not exist or does not define a processNode value.

Example

```
techGetProcessNode(tech)=> 0.02
```

Returns `0.02` as the processNode value for the technology database `tech`.

techGetViewTypeUnits

```
techGetViewTypeUnits(
     d_techFileID
     )
     => l_units / nil
```

Description

Gets the units that are set for the each cellview type in the specified technology database.

Arguments

d_techFileID	The database identifier of the technology file.

Value Returned

l_units	A list of the he units that are set for the each cellview type in the specified technology file.
nil	The technology file does not exist or you do not have read permission on the technology file.

Example

```
techGetViewTypeUnits(tfID)
=> (("maskLayout" "micron" 1000)
       ("schematic" "inch" 160)
       ("schematicSymbol" "inch" 160)
       ("netlist" "inch" 160)
)
```

Returns the user units for each of the cellview types (`maskLayout`, `schematic`, `schematicSymbol`, and `netlist`) in the technology file with the database identifier stored in `tfID`.

techIsMfgGridResolutionSet

```
techIsMfgGridResolutionSet(
     d_techID
     )
     => t / nil
```

Description

Checks whether the value used for grid snapping (the default manufacturing grid resolution for design sessions) is set in the specified technology database. ASCII technology file location: `mfgGridResolution` subsection in the `controls` section.

For more information about the `mfgGridResolution` section of the technology file, see mfgGridResolution.

Arguments

d_techID	The identifier of the technology database.

Value Returned

`t`	The manufacturing grid resolution was updated or created successfully.
`nil`	The technology database does not exist.

Example

```
techIsMfgGridResolutionSet(tfID)
=> t
```

The manufacturing grid resolution value in the technology database identified by `tfID` is set.

techIsReadPermission

```
techIsReadPermission(
     d_techFileID
     t_sectionname
     )
     => t / nil
```

Description

Indicates whether the current user has read permission explicitly set on the specified section in the specified technology database.

Arguments

d_techFileId	The database identifier of the technology file.
t_sectionname	The name of the technology file section.

Value Returned

`t`	The user has read permission explicitly set.
`nil`	The user does not have read permission explicitly set or the technology file does not exist.

Example

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

Checks for read permission for the current user on the `layerDefinitions` section in the technology file with the database identifier stored in `tfID`; it displays `t` to indicate that the user has read permission explicitly set.

techSetFabricType

```
techSetFabricType(
     d_techfileID
     t_fabricType
     )
     => t / nil
```

Description

Sets the fabric type for a technology database.

For more information about the `fabricType` section of the technology file, see fabricType.

Arguments

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

Value Returned

`t`	The specified fabric type has been set for the technology database.
`nil`	The fabric type could not be set, or the technology database does not exist.

Example

```
techSetFabricType(tech "ic")
=> t
```

Sets `ic` as the fabric type for the technology database.

techSetMfgGridResolution

```
techSetMfgGridResolution(
     d_techID
     g_resolution
     )
     => t / nil
```

Description

Updates or sets the value used for grid snapping in the specified technology database. ASCII technology file location: `mfgGridResolution` subsection in the `controls` section.

For more information about the `mfgGridResolution` section of the technology file, see mfgGridResolution.

Arguments

d_techID	The identifier of the technology database.
g_resolution	The number of user units to use as the basis for the grid. Valid values: Any floating-point number, any integer

Value Returned

`t`	The manufacturing grid resolution was updated or created successfully.
`nil`	The technology file does not exist.

Example

```
techSetMfgGridResolution(tfID 0.002)
=> t
```

Sets the manufacturing grid resolution value to `0.002` in the technology file with the database identifier stored in `tfID`.

techSetParam

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

Description

Updates the value of the specified control parameter in the specified technology database. 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

Value Returned

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

Example

```
tfID = techGetTechFile(ddGetObj("testLib"))
db:0x0180200d
techSetParam(tfID "lambda" 0.65)
=> t
```

Updates the value of the control parameter `lambda` in the technology file identified by `tfID` (`testLib`).

techSetReadPermission

```
techSetReadPermission(
     d_techFileID
     t_sectionname
     t_rUser
     )
     => t / nil
```

Description

Sets read permission on the specified section in the specified technology database for the specified user.

Arguments

d_techFileId	The database identifier of the technology file.
t_sectionname	The name of the technology file section.
t_rUser	The name of the user.

Value Returned

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

Example

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

Sets read-only permission for user `joe` on the `layerDefinitions` section in the technology file with the database identifier stored in `tfID`.

techSetReadWritePermission

```
techSetReadWritePermission(
     d_techFileID
     t_sectionname
     t_rwUser
     )
     => t / nil
```

Description

Sets read/write permission on the specified section in the specified technology database for the specified user.

Arguments

d_techFileId	The database identifier of the technology file.
t_sectionname	The name of the technology file section.
t_rwUser	The name of the user.

Value Returned

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

Example

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

Sets read/write permission for user `joe` on the `layerDefinitions` section in the technology file with the data base identifier stored in `tfID`.

3

Layers Functions

The layer functions let do the following:

- Add or update sections in the technology database.
- Add or update to sections in the technology database.
- Retrieve data from the technology database.
- Delete data from the technology database.

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

analysisAttributeLocks

```
analysisAttributeLocks(
     l_list
     )
     => t / nil
```

Description

Specifies the technology class to define layer analysis attribute lock.

Arguments

l_list	List of layer analysis attribute locks.

Value Returned

t	The function successfully specifies the technology class to define layer analysis attribute lock.
nil	The function does not specify the technology class to define layer analysis attribute lock.

Example

```
analysisAttributeLocks(
    (lockAddLayers        t)
    (lockValuesChange     t)
)
```

techCreateDerivedLayer

```
techCreateDerivedLayer(
     d_techID
     x_derivedLayerNum
     t_derivedLayerName
     tx_layer1
     t_op
     [ tx_layer2 ]
     [ l_options ]
     )
     => d_techLayerID / nil
```

Description

Creates a derived layer in the specified technology database.

Some layer operations can use parameters to determine how a layer is derived. The following table lists such operators and the parameters they support.

Operators	Parameters
select	selectShapesWithPurpose
area	areaRange
grow, growVertical, growHorizontal, shrink, shrinkVertical, shrinkHorizontal	distance
inside	connectivityType, range, selectShapesInRange
buttOnly	range, selectShapesInRange, exclusive
overlapping, straddling, coincident, coincidentOnly,enclosing, butting, buttingOrCoincident, buttingOrOverlapping	connectivityType, range, selectShapesInRange, exclusive
color	maskColor, colorLocked

Arguments

d_techID	The database identifier of the technology database.
t_derivedLayerName	The name of the derived layer to be created. Valid values: Any string
x_derivedLayerNum	The number of the derived layer to be created. Valid values: Any positive integer that is not a DFII reserved layer number
tx_layer1	The first or only layer used to create the derived layer. Valid values: The layer name or layer number
t_op	The operation on the layer or layers that creates the derived layer. Valid values: ■ One-layer derived layers ❑ Purpose-aware derived layers: `select` ❑ Sized derived layers: `grow`, `shrink`, `growVertical`, `growHorizontal`, `shrinkVertical`, `shrinkHorizontal` ❑ Area-restricted derived layers: `area` ❑ Color derived layers: `color` ■ Two-layer derived layers: `and`, `or`, `not`, `xor`, `touching`, `buttOnly`, `inside`, `outside`, `overlapping`, `straddling`, `avoiding`, `butting`, `coincident`, `coincidentOnly`, `enclosing`, `buttingOrCoincident`, `buttingOrOverlapping` For more information about these operators, see techDerivedLayers in *Virtuoso Technology Data ASCII Files Reference*.
tx_layer2	The second layer used to create the derived layer. This layer is required for derived layers that operate on two layers. Valid values: The layer name or layer number

l_options

A list specifying options. It has the following syntax:

```
list(
     [ list("connectivityType"
                    { "diffNet" | "sameNet" })
     ]
     [ list("exclusive" { t | nil }) ]
     [ list("range" t_rangeVal) ]
     [ list("areaRange" t_areaRangeVal) ]
     [ list("selectShapesInRange" { t | nil }) ]
     [ list("distance" x_distValue | f_distValue) ]
     [ list("selectShapesWithPurpose" tx_purpose) ]
     [ list("maskColor" t_maskColor)
       [ list("colorLocked" { "locked" | "unlocked" }) ]
     ]

)
```

where,

- `connectivityType`: Controls whether selected shapes must connect to different nets (`diffNet`) or to the same net (`sameNet`). If a value is not specified, connectivity is ignored.
- `exclusive`: Selects *`tx_layer1`* shapes only if these shapes have no relationship other than that specified by the layer operation this parameter modifies.
  If set to `nil`, shapes are selected even if there are other shapes on *`tx_layer1`* interacting in other ways with the shapes on *`tx_layer2`*.
- `range`: Selects shapes only if the number of shapes under consideration falls in the specified range. One or two values are required depending on the specified range type.
  Valid values for range type: *n*, <*n*, <=*n*, >*n*, >=*n*, [*n1 n2*], (*n1 n2*), [*n1 n2*), (*n1 n2*]

  where,
  n is a count, *n1* is the lower bound, *n2* is the upper bound, `[ ]` indicates an inclusive range, and `( )` indicates an exclusive range

- `selectShapesInRange`: Determines if the number of shapes should be inside or outside the specified range. If set to `t`, shapes are selected only if the specified range value is satisfied, which is also the default behavior; otherwise, shapes are selected even if their number falls outside the specified range.
- `areaRange`: Indicates the area range to be used for the `area` operation. The value *`t_areaRangeVal`* specifies the value and type of range—whether the range is a lower bound only, an upper bound only, or both a lower and upper bound. One or two values are required depending on the range type specified.
- `selectShapesWithPurpose`: Specifies a purpose value for selecting shapes. It is used as a parameter by the `select` layer operation.
- `distance`: Specifies the distance for layer sizing operations.
- `maskColor`: Specifies the mask value for the shapes to be selected, such as `mask1Color`, `mask2Color`, and so on. The specified mask value must be supported by the layer to which the `color` operator is applied.
- `colorLocked`: Specifies the lock state for the shapes to be selected.
  Valid values: `locked`, `unlocked`, `any`

Value Returned

`d_techLayerID`	The derived layer was created successfully with the database identifier *`techLayerID`*.
`nil`	The technology database does not exist or the derived layer could not be created.

Examples

```
techCreateDerivedLayer(tfID 20000 "nwellBlockGen" "nwell" "select"
list(list("selectShapesWithPurpose" "blockGen")))
=> db:0x189fe736
```

Creates a derived layer by selecting layer `"nwell"` and purpose `"blockGen"`.

```
techCreateDerivedLayer(tfID 30001 "myDL1" "M1" "or" "M2")
=> db:0x01d0700e
```

Creates a derived layer by selecting shapes that correspond to a union of the shapes on `metal1` and `metal2`.

```
techCreateDerivedLayer(tfID 30011 "myDL11" "M1" "grow" list(list("distance" 0.1)))
=> db:0x186ee62c
techCreateDerivedLayer(tfID 30012 "myDL12" "M1" "growVertical"
list(list("distance" 0.1)))
=> db:0x186ee62e
techCreateDerivedLayer(tfID 30013 "myDL13" "M1" "growHorizontal"
list(list("distance" 0.1)))
=> db:0x186ee62f
techCreateDerivedLayer(tfID 30014 "myDL14" "M1" "shrink"
list(list("distance" 0.1)))
=> db:0x1792ed23
techCreateDerivedLayer(tfID 30015 "myDL15" "M1" "shrinkVertical"
list(list("distance" 0.1)))
=> db:0x1792ed25
techCreateDerivedLayer(tfID 30016 "myDL16" "M1" "shrinkHorizontal"
list(list("distance" 0.1)))
=> db:0x186ee633
```

Creates derived layers by sizing `M1`.

```
techCreateDerivedLayer(tfID 30003 "myDL3" "M1" "xor" "M2")
=> db:0x1792ed23
```

Creates a derived layer by selecting non-overlapping areas of shapes on `M1` and `M2`.

```
techCreateDerivedLayer(tfID 30004 "myDL4" "M1" "area"
list(list("areaRange" "< 8")))
=> db:0x1792ed26
```

Creates a derived layer with restricted area.

```
techCreateDerivedLayer(tfID 30005 "myDL5" "M1" "and" "M2")
=> db:0x1792ed24
```

Creates a derived layer by selecting shapes that correspond to the intersecting areas on `M1` and `M2`.

```
techCreateDerivedLayer(tfID 30006 "myDL6" "M2" "inside" "M3"
list(list("connectivityType" "sameNet") list("range" 9)
list("selectShapesInRange" t)))
=> db:0x1792ed27
```

Creates a derived layer by selecting `M2` shapes that are completely inside a shape on `M3`, provided the number of such shapes is equal to `9`.

```
techCreateDerivedLayer(tfID 30007 "myDL7" "M1" "buttOnly" "M2"
list(list("range" "(2 3)") list("selectShapesInRange" t) list("exclusive" t)))
=> db:0x18cae7a9
```

Creates a two-layer derived layer by selecting `M1` shapes that satisfy the `buttOnly` condition, provided that the shapes exclusively satisfy this condition and the number of such shapes is equal to `2` or `3`.

```
techCreateDerivedLayer(tfID 30021 "myDL21" "M1" "overlapping" "M2"
list(list("connectivityType" "sameNet") list("range" "(2 3)")
list("selectShapesInRange" t) list("exclusive" t)))
=> db:0x1792ed27

techCreateDerivedLayer(tfID 30022 "myDL22" "M1" "straddling" "M2"
list(list("connectivityType" "sameNet") list("range" "(2 3)")
list("selectShapesInRange" t) list("exclusive" t)))
=> db:0x186ee62f

techCreateDerivedLayer(tfID 30023 "myDL23" "M1" "coincident" "M2"
list(list("connectivityType" "sameNet") list("range" "(2 3)")
list("selectShapesInRange" t) list("exclusive" t)))
=> db:0x1792ed24

techCreateDerivedLayer(tfID 30024 "myDL24" "M1" "coincidentOnly" "M2"
list(list("connectivityType" "sameNet") list("range" "(2 3)")
list("selectShapesInRange" t) list("exclusive" t)))
=> db:0x1792ed26

techCreateDerivedLayer(tfID 30025 "myDL25" "M1" "enclosing" "M2"
list(list("connectivityType" "sameNet") list("range" "(2 3)")
list("selectShapesInRange" t) list("exclusive" t)))
=> db:0x1792ed29

techCreateDerivedLayer(tfID 30026 "myDL26" "M1" "butting" "M2"
list(list("connectivityType" "sameNet") list("range" "(2 3)")
list("selectShapesInRange" t) list("exclusive" t)))
=> db:0x186ee630

techCreateDerivedLayer(tfID 30027 "myDL27" "M1" "buttingOrCoincident" "M2"
list(list("connectivityType" "sameNet") list("range" "(2 3)")
list("selectShapesInRange" t) list("exclusive" t)))
=> db:0x186ee631

techCreateDerivedLayer(tfID 30028 "myDL28" "M1" "buttingOrOverlapping" "M2"
list(list("connectivityType" "sameNet") list("range" "(2 3)")
list("selectShapesInRange" t) list("exclusive" t)))
=> db:0x186ee62f
```

Creates two-layer derived layers by selecting `M1` shapes that satisfy the condition imposed by the specified operator, provided that the shapes exclusively satisfy this condition and the number of such shapes is equal to `2` or `3`.

```
techCreateDerivedLayer(tfID 30009 "metal1Color1" "M1" "color"
list(list("maskColor" "mask1Color")))
=> db:0x1792ed2a
```

Creates a derived layer by copying to it `M1` shapes with `mask1Color`.

```
techCreateDerivedLayer(tfID 30010 "metal1Color2" "M1" "color"
list(list("maskColor" "mask1Color") list("colorLocked" "locked")))
=> db:0x1792ed2b
```

Creates a derived layer by copying to it `M1` shapes that have `mask1Color` locked on them.

techCreateLayer

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

Description

Creates a layer in the specified technology database. ASCII technology file location: `techLayers` subsection in the `layerDefinitions` section; it lists the layers that can be used. If a `techLayers` subsection 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` section, see techLayers in the *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
x_layerNumber	The layer number. Valid values: A unique integer from `0` through `194` and from `256` through $2^{31}-1$ (layers numbered `195` through `255` are system-reserved layers)
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

Value Returned

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

Example

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

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

techCreateLP

```
techCreateLP(
     d_techID
     l_layerPurpose
     t_lpName
     )
     => d_lpID / nil
```

Description

Creates a named layer-purpose pair in the specified technology database. Layer-purpose pairs are used to define how layers are displayed. A layer is paired with a purpose, assigned a display packet and display attributes (in the `techDisplays` subsection in the `layerDefinitions` section of the ASCII technology file).

`techCreateLP` defines the layer-purpose pair with defaults in the `techLayerPurposePriorities` and `techDisplays` subsections of the `layerDefinitions` section.

This function returns a database identifier for the layer-purpose pair. 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 database before you can create a layer-purpose pair.

For more information about the `techLayerPurposePriorities` subsection of the technology file, see techLayerPurposePriorities in *Virtuoso Technology Data ASCII Files Reference*. For more information about the `techDisplays` subsection of the technology file, see techDisplays in *Virtuoso Technology Data ASCII Files Reference*.

The defaults set by this function are as follows:

Technology File Subsection	Argument	Default
`techLayerPurposePriorities`	Priority	`0`

Technology File Subsection	Argument	Default
techDisplays	Packet	defaultPacket
	Visibility	t
	Selectability	t
	Contributes to changed layer	t
	Drag enable	t
	Valid	t

Arguments

d_techID	The database identifier of the technology database.
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. Valid values: Must be the same name as the layer name

Value Returned

d_lpID	The database identifier for the layer-purpose pair.
nil	The technology database does not exist or the layer or purpose is not defined in the technology database; the layer-purpose pair is not created.

Example

```
techCreateLP(tfID("nwell" "drawing") "nwell")
=> db:18483792
```

Creates a layer-purpose pair named newell with the nwell layer and the drawing purpose in the technology database identified by tfID. Returns the database identifier. Also updates technology subsections in the database with information about the new layer-purpose pair as illustrated by the ASCII syntax below:

```
techLayerPurposePriorities(
     ( nwell drawing)
           .
           .
           .
     )

techDisplays
     ( nwell drawing defaultPacket t t t t t)
           .
           .
           .
     )
```

techCreatePurpose

```
techCreatePurpose(
     d_techID
     x_purposeNumber
     t_purposeName
     [ t_purposeAbbrev ]
     )
     => t / nil
```

Description

Creates a purpose in the specified technology database. ASCII technology file location: `techPurposes` subsection in the `layerDefinitions` section; it lists the layer purposes used with technology data. If a `techPurposes` subsection does not exist, this function creates one with the specified data.

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 `techPurposes`, see techPurposes in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
x_purposeNumber	The purpose number.
	Valid values: A unique integer, `1` through `128` and `256` through 2^{32}-65535.
t_purposeName	The purpose name.
	Valid values: Any string
t_purposeAbbrev	The optional purpose abbreviation.
	Valid values: Any string of seven characters or less

Value Returned

`t`	The purpose was successfully created.
`nil`	The technology database does not exist; the purpose is not created.

Example

```
techCreatePurpose(techID 10 "drawing" "dwg")
=> t
```

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

techCreatePurposeDef

```
techCreatePurposeDef(
     d_techfileId
     x_purposeNumber
     t_purposeName
     [ t_purposeAbbrev ]
     )
     => d_purposeDefId / nil
```

Description

Creates a purpose in the specified technology database. ASCII technology file location: `techPurposes` subsection in the `layerDefinitions` section; it lists the layer purposes used with technology data. If a `techPurposes` subsection does not exist, this function creates one with the specified data.

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 `techPurposes`, see techPurposes in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
x_purposeNumber	The purpose number. Valid values: A unique integer, `1` through `128`, plus `256` through $2^{32}-1026$
t_purposeName	The purpose name. Valid Values: Any string
t_purposeAbbrev	The optional purpose abbreviation. Valid values: Any string of seven characters or less

Value Returned

d_purposeDefId	The purpose object ID.
`nil`	The technology database does not exist; the purpose is not created.

Example

```
p1101 = techCreatePurposeDef(tfId 1101 "purpose101" "P01")
```

Creates the `purposeDefId`, `p1101` in the specified technology database.

techDeleteLayer

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

Description

Deletes the specified layer from the specified technology database. ASCII technology file location: `techLayers` subsection in the `layerDefinitions` section; it lists the layers that can be used.

Caution

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

For more information about `techLayers`, see techLayers in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The name or number of the layer to delete.

Value Returned

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

Example

```
techDeleteLayer(tfID "metal5")
```

Deletes the `metal5` layer from the technology database identified by `tfID`.

techDeleteLP

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

Description

Deletes the specified layer-purpose pair from the current technology database. ASCII technology file location: in the `techLayerPurposePriorities` and `techDisplays` subsections of the `layerDefinitions` section.

This function only deletes the layer-purpose–pair definition from the layerDefinitions in the technology database. If the layer-purpose pair is referenced in other sections of the database, 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 reload it.

For more information about the `layerDefinitions` section, see Technology File Layer Definitions in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

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

Value Returned

`t`	The layer-purpose pair was deleted from the `layerDefinitions` section.
`nil`	The technology database does not exist or the layer-purpose pair is not defined in the technology database.

Example

```
techDeleteLP(lpID)
=> t
```

Deletes the layer-purpose pair identified by `lpID` from the `layerDefinitions` section. References to the layer-purpose pair might still exist in other technology database sections, which must be updated separately.

techDeletePurpose

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

Description

Deletes the specified purpose from the specified technology database. ASCII technology file location: `techPurposes` subsection in the `layerDefinitions` section; it lists the layer purposes used with technology data.

This function does not purge all references of the purpose from the technology database. If the purpose is referenced in another section of the technology database, an error will occur when an application attempts to use that data. To prevent these errors, you must delete all references to the purpose from the ASCII technology file and recompile it.

For more information about `techPurposes`, see techPurposes in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
tx_purpose	The purpose name or number.

Value Returned

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

Example

```
techDeletePurpose(tfID 13)
=> t
```

Deletes the purpose number `13` from the technology database identified by `tfID`.

```
techDeletePurpose(tfID "drawing5")
=> t
```

Deletes the purpose `drawing5` from the technology database identified by `tfID`.

techDeletePurposeDef

```
techDeletePurposeDef(
     d_purposeDefId
     )
     => t / nil
```

Description

Deletes the specified purpose from the specified technology database. ASCII technology file location: `techPurposes` subsection in the `layerDefinitions` section; it lists the layer purposes used with technology data.

This function does not purge all references of the purpose from the technology database. If the purpose is referenced in another section of the technology database, an error will occur when an application attempts to use that data. To prevent these errors, you must delete all references to the purpose from the ASCII technology file and recompile it.

For more information about `techPurposes`, see techPurposes in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_purposeDefId	The purposeDef ID.

Value Returned

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

Example

```
techDeletePurposeDef(p1101)
```

Deletes the `purposeDefId`, `p1101` from the specified technology database.

techDeleteTwoLayerProp

```
techDeleteTwoLayerProp(
     d_techID
     tx_layer1
     tx_layer2
     t_name
     )
     => t / nil
```

Description

Deletes the specified two-layer property from the specified technology database. ASCII technology file location: `techLayerProperties` subsection in the `layerDefinitions` section; it specifies special properties that you want to place on the layers in your design.

For more information about `techLayerProperties`, see techLayerProperties in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
tx_layer1	The first layer.
	Valid values: The layer name or number or a list containing the layer name and purpose
tx_layer2	The second layer.
	Valid values: The layer name or number or a list containing the layer name and purpose
t_name	The name of the two-layer property to delete.

Value Returned

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

Example

```
techDeleteTwoLayerProp(tfID "metal" "via" "prop1")
=> t
```

Deletes the `prop1` property assigned to the `metal` and `via` layers in the technology database identified by `tfID`.

techFindLayer

```
techFindLayer(
     d_techfileID
     tx_layer
     )
     => d_layerID / nil
```

Description

Returns the layerID in the specified technology database or a referenced technology database in an ITDB graph of the specified technology database. This function returns the layerID as per the specified layer name or layer number.

Arguments

d_techfileID	The database identifier of the technology database.
tx_layer	The layer name or number.

Value Returned

d_layerID	The layerID is returned in the specified technology database.
nil	Unable to find layer ID.

Example

```
layer2 = techFindLayer(tf "layer2")
```

Returns the layer ID based on database identifier, `tf` and layer, `layer2`.

techFindPurposeDef

```
techFindPurposeDef(
     d_techfileID
     tx_purpose
     )
     => d_purposeDefID / nil
```

Description

Returns the purposeDefID in the specified technology database or a referenced technology database in an ITDB graph of the specified technology database. This function returns the purposeDefID as per the specified purpose name or purpose number.

Arguments

d_techfileID	The database identifier of the technology database.
tx_purpose	The purpose name or number.

Value Returned

d_purposeDefID	The purposeDefID is returned in the specified technology database.
nil	Unable to find PurposeDef ID.

Example

```
drawing = techFindPurposeDef(tf "drawing")
```

Returns the purposeDefID based on database identifier, `tf`, and purpose, `drawing`.

techGetDerivedLayer

```
techGetDerivedLayer(
     d_techID
     tx_layer1
     t_op
     tx_layer2
     )
     => d_techLayerID / nil
```

Description

Returns the database identifier for the derived layer created from the two specified layers with the specified operation in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
tx_layer1	Is used to create the derived layer. Valid values: The layer name or layer number
t_op	The operation between *layer1* and *layer2* that produces the derived layer. Valid values: `and`, `or`, `not`, `xor`, `touching`, `buttOnly`
tx_layer2	Is used to create the derived layer. Valid values: The layer name or layer number

Value Returned

d_techLayerID	The database identifier of the derived layer.
`nil`	The technology database does not exist or the derived layer does not exist.

Example

```
techGetDerivedLayer(tfID "metal1" "or" "metal2" )
=> db:0x01d0700e
```

Returns the database identifier for the derived layer defined as `metal1 or metal2`.

techGetEquivLayers

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

Description

Returns a list of the equivalent layers defined in the specified technology database. ASCII technology file location: `equivalentLayers` subsection in the `layerRules` section; it lists layers that represent the same kind of material.

For more information about the `equivalentLayers` subsection of the technology file, see equivalentLayers in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_equivLayersList

List of lists indicating which layers are equivalent. The list has the following syntax:

`( ( lt_layer … ) … )`

where, *lt_layer* is an equivalent layer in the `equivalentLayers` subsection of the technology database. The layer is listed as it appears in the technology database. It can be a layer name or a layer-purpose pair.

`nil`	The technology database does not exist or does not define any equivalent layers.

Example

```
techGetEquivLayers(tfID)
=> (("metal1" "metal2")
      ("via1" "via2" "via3")
      (("metal3" "pin") "pinMetal")
)
```

Returns the equivalent layers defined in the technology database identified by `tfID`.

techGetLabelLayers

```
techGetLabelLayers(
     d_techID
     )
     => l_labelLayers / nil
```

Description

Returns a list of the label layers defined in the specified technology database. ASCII technology file location: `labels` subsection in the `layerRules` section.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_labelLayers	A list of the label layers.
`nil`	The technology database does not exist or no `labels` subsection is specified in the technology database.

techGetLayerAbbrev

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

Description

Returns the abbreviation of the specified layer from the specified technology database. ASCII technology file location: `techLayers` subsection in the `layerDefinitions` section; it lists the layers that can be used.

For more information about `techLayers`, see techLayers in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The layer name or number.

Value Returned

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

Example

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

Returns the abbreviation of layer number `16` defined in the technology database identified by `tfID`.

```
techGetLayerAbbrev(tfID "metal2")
=> "met2"
```

Returns the abbreviation of the `metal2` layer defined in the technology database identified by `tfID`.

techGetLayerAnalysisAttribute

```
techGetLayerAnalysisAttribute(
     d_techfileID
     tx_layer
     t_attributeName
     )
     => g_attributeValue / nil
```

Description

Returns the value of a specified attribute for a layer with a layer function that supports attribute analysis.

Arguments

d_techfileID	The database identifier of the technology database.
tx_layer	The layer name or number.
t_attributeName	The name of the attribute. The valid attribute names are: ■ `materialName`: The name of the material. ■ `thickness`: The thickness of the layer in microns. ■ `conductivity`: The conductivity of the layer in siemens/meter. ■ `permittivity`: The dielectric constant of the layer. ■ `lossTangent`: The loss tangent of the layer.

Value Returned

g_attributeValue	The value of the specified attribute.
`nil`	The specified attribute is not set on the layer, or the layer does not have a function that supports layer analysis attributes or does not exist in the technology database.

Example

```
techGetLayerAnalysisAttribute(tech "Metal1" "lossTangent")
=> 0.9
```

Returns `0.9` as the value of the `lossTangent` attribute for the `Metal1` layer.

techGetLayerAnalysisAttributeLock

```
techGetLayerAnalysisAttributeLock(
     d_techID
     s_analysisAttributeLockType
     )
     => t / nil
```

Description

Returns the state of the lock on layer analysis attributes for changes related to addition of new layers or updates of attributes.

The following analysis attribute locks are supported for this function:

- `lockAddLayers`: Prevents addition or removal of layers from the stack but their values can be changed.
- `lockValuesChange`: Prevents change of layer attributes. However, layers can be added or removed.

Arguments

d_techID	The database identifier of the technology database.
s_analysisAttributeLockType	The type of analysis attribute lock. Valid values: `lockAddLayers`, `lockValuesChange`.

Value Returned

`t`	The lock is active.
`nil`	The lock is not active.

Example

```
techID = techGetTechFile(geGetEditCellView())
lockType = "lockAddLayers"
when(techGetLayerAnalysisAttributeLock(techID lockType)
    print("'Add layers' lock is currently active")
)
```

techGetLayerFunction

```
techGetLayerFunction(
     d_techID
     tx_layer
     )
     => g_function / nil
```

Description

Returns the function (material) assigned to the specified layer in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The layer name or layer number or a list containing the layer name and purpose.

Value Returned

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

Example

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

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

techGetLayerFunctions

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

Description

Returns a list of the functions (materials) and mask numbers assigned to the user-defined layers in the specified technology database. Returns function unknown for layers without an assigned function.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

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

Example

```
techGetLayerFunctions(tfID)
=> (("Oxide" "poly" 1)
       ("Nwell" "nwell" 2)
       ("Poly" "poly" 3)
       ("Nimp" "ndiff" 4)
       …
       ("Metal5_slot" "unknown" 94)
)
```

Returns a list of the user-defined layers in the technology database identified by `tfID` along with the functions and mask numbers assigned to them; returns `unknown` for layers without functions assigned to them.

techGetLayerMaskNumber

```
techGetLayerMaskNumber(
     d_techID
     tx_layer
     )
     => x_maskNumber / nil
```

Description

Returns the mask number assigned to the specified layer in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The layer name or number or a list containing the layer name and purpose.

Value Returned

x_maskNumber	The mask number assigned to the specified layer.
`nil`	The technology database does not exist, there is no `functions` definition for the specified layer, or there is no mask number assigned to the specified layer.

Example

```
techGetLayerMaskNumber(tfID "metal1")
=> 1
```

Returns the mask number, `1`, assigned to the layer `metal1` in the technology database identified by `tfID`.

techGetLayerMfgResolution

```
techGetLayerMfgResolution(
     d_techID
     tx_layer
     )
     => g_value / g_mfgGridResolution / nil
```

Description

Returns the manufacturing grid resolution assigned to the specified layer in the specified technology database. ASCII technology file location: `layerRules` section in the specified technology file.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The layer name or layer number.

Value Returned

g_value	The manufacturing grid resolution applied to the layer.
g_mfgGridResolution	The default manufacturing grid resolution specified in the `mfgGridResolution` subsection of the `controls` section. This value is returned if no layer manufacturing grid resolution is specifically assigned to the layer.
`nil`	The technology database does not exist, the layer is not defined in the technology database, or no manufacturing grid resolution is specified in the technology database, either with `mfgResolutions` in the `layerRules` section or `mfgGridResolution` in the `controls` section.

Example

```
techGetLayerMfgResolution(tfID "metal1")
=> 0.002
```

Returns the manufacturing grid resolution, `0.002`, assigned to be applied to the layer `metal1` in the technology database identified by `tfID`.

techGetLayerMfgResolutions

```
techGetLayerMfgResolutions(
     d_techID
     )
     => l_layers
```

Description

Returns each layer assigned a layer manufacturing grid resolution and the layer manufacturing grid resolution applied to it in the specified technology database. ASCII technology file location: `mfgResolutions` subsection in the `layerRules` section.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_layers	A list of lists containing the layer and manufacturing grid resolution data. The list has the following syntax: `list ( ( tx_layer g_value ) … )` where, ■ *tx_layer* is the layer name. ■ *g_value* is the manufacturing grid resolution assigned to apply to the layer.

Example

```
techGetLayerMfgResolutions(tfID)
=> (("metal1" 0.002)
       ("metal2" 0.003)
)
```

Returns the layer manufacturing grid resolution assigned to be applied to layers in the technology database identified by `tfID`: `0.002` to `metal1` and `0.003` to `metal2`.

techGetLayerName

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

Description

Returns the name of the layer associated with the specified layer number defined in the specified technology database. ASCII technology file location: `techLayers` subsection in the `layerDefinitions` section; it lists the layers that can be used.

For more information about `techLayers`, see techLayers in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
x_layerNumber	The layer number. Valid values: An integer from `0` through $2^{32}-1$. **Note:** Layers numbered from `195` through `255` are reserved for the system, as are layers numbered from $2^{32}-1025$ through $2^{32}-1$.

Value Returned

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

Example

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

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

techGetLayerNum

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

Description

Returns the layer number associated with the specified layer name defined in the specified technology database. ASCII technology file location: `techLayers` subsection in the `layerDefinitions` section; it lists the layers that can be used.

For more information about `techLayers`, see techLayers in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
t_layerName	The layer name.

Value Returned

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

Example

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

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

techGetLayerProp

```
techGetLayerProp(
     d_techID
     tx_layer
     t_propName
     )
     => g_propValue / nil
```

Description

Returns the value of the specified layer property from the specified technology database. ASCII technology file location: `techLayerProperties` subsection in the `layerDefinitions` section; it specifies special properties that you want to place on the layers in your design.

For more information about `techLayerProperties`, see techLayerProperties in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The layer. Valid values: The layer name or number or a list containing the layer name and purpose.
t_propName	The name of the property.

Value Returned

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

Example

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

Returns `well`, the value of the layer property `myProp` assigned to the layer-purpose pair `nwell drawing` in the technology database identified by `tfID`.

techGetLayerRoutingDirections

```
techGetLayerRoutingDirections(
     d_techID
     )
     => l_layerRoutingDirections / nil
```

Description

Returns a list of the layer routing directions for the routing layers in the specified technology database. ASCII technology file location: `routingDirections` subsection in the `layerRules` section.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_layerRoutingDirections

A list of the layer routing directions specified for the routing layers in the technology database.

`nil` The technology database does not exist or no routing directions are specified on the routing layers in the technology database.

Example

```
techGetLayerRoutingDirections (tfID)
=> (("metal1" "horizontal")
    ("metal2" "vertical")
    ("metal3" "leftDiag")
    ("metal4" "rightDiag")
   )
```

Returns a list of the routing directions specified for the routing layers in the technology database identified by `tfID`.

echGetLayerRoutingGrid

```
techGetLayerRoutingGrid(
     d_techID
     tx_layer
     )
     => l_layerRoutingGrid / nil
```

Description

Returns the layer routing direction, pitch, and offset for the specified routing layer in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The name or number of the layer for which to get the routing definition.

Value Returned

l_layerRoutingGrid

A list containing the routing definition. The list has the following syntax:

```
( t_preferredDir
[ g_pitch [ g_offset ] ] )
```

where,

- *t_preferredDir* is the keyword used to indicate how place-and-route software to use the layer.
  Valid Values: `none`, `horizontal`, `vertical`, `leftDiag`, `rightDiag`
- *g_pitch* is the pitch, in user units, for the routing grid of the layer.
- *g_offset* is the offset, in user units, for the routing grid of the layer.

`nil` — The technology database does not exist, the specified layer is not defined, or the specified layer has no routing definition assigned.

Example

```
techGetLayerRoutingGrid(tfID "metal1")
=> (“horizontal” 0.51)
```

Returns the routing definition specified in the current technology database, identified by `tfID`, for the layer `metal1`. The layer is assigned the direction `horizontal` and a pitch of `0.51`.

techGetLayerRoutingGrids

```
techGetLayerRoutingGrids(
     d_techID
     )
     => l_layerRoutingGrids / nil
```

Description

Returns the layer routing direction, pitch, and offset assigned to layers in the specified technology database. ASCII technology file location: `routingDirections` subsection in the `layerRules` section.

Arguments

d_techID	The database identifier of the technology database.
t_cvType	The name of the cellview type. Valid values: String specifying a valid cellview type (for example, `maskLayout`)

Value Returned

l_layerRoutingGrids

A list of routing definitions. The list has the following syntax:

```
( ( t_layer t_preferredDir [ g_pitch
[ g_offset ] ] ) … )
```

where,

- *t_layer* is the name of the routing layer.
- *t_preferredDir* is the keyword used to indicate how the place-and-route software is to use the layer.

  Valid values: `none`, `horizontal`, `vertical`, `leftDiag`, `rightDiag`
- *g_pitch* is the pitch, in user units, for the routing grid of the layer as specified in the foundry constraint group.
- *g_offset* is the offset, in user units, for the routing grid of the layer as specified in the foundry constraint group.

Note: The function cannot be used to retrieve the pitch and offset set in any other constraint group.

`nil` — The technology database does not exist or no layer routing grids are defined in the specified technology database.

Example

```
techGetLayerRoutingGrids(tfID)
=> (("METAL1" "horizontal" 0.51)
       ("METAL2" "vertical" 0.51)
       ("METAL3" "horizontal" 0.51)
       ("METAL4" "vertical" 0.51)
       ("METAL5" "horizontal" 0.51)
       ("METAL6" "vertical" 0.51)
       ("METAL7" "horizontal" 1.02)
       ("METAL8" "vertical" 1.02)
)
```

Returns the layer routing data specified in the current technology database, identified by `tfID`. Each layer is assigned a direction and pitch.

techGetLP

```
techGetLP(
     d_techID
     l_layerPurpose
     )
     => l_layerPurposeID / nil
```

Description

Returns the database identifier of the specified layer-purpose pair specified in the current technology database.

Arguments

d_techID	The database identifier of the technology database.
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 )`

Value Returned

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

Example

```
techGetLP(tfID list("diff" "drawing"))
=> db:18483790
```

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

techGetLPAttr

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

Description

Returns the list of attributes for the specified layer-purpose pair in the current technology database. The attributes (display priority, visibility, selectability, if the layer is visible when dragged, and validity) are defined in the `techLayerPurposePriorities` and `techDisplays` subsections of the ASCII 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` subsection of the technology file, see techLayerPurposePriorities in *Virtuoso Technology Data ASCII Files Reference*. For more information about the `techDisplays` subsection of the technology file, see techDisplays in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_layerPurposeID	The database identifier of the layer-purpose pair.

Value Returned

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 )` where, ■ *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. ■ *g_dragEnable* indicates whether you can drag a shape created with the layer-purpose pair in the layout editor. ■ *g_valid* indicates whether the layer-purpose pair appears in the LSW.
`nil`	The technology database or layer-purpose pair does not exist.

Example

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

Returns the attributes assigned to the layer-purpose pair identified by `lpID`.

techGetLPPacketName

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

Description

Returns the name of the display packet defined for the specified layer-purpose pair in the current technology database. ASCII technology file location: `techDisplays` subsection of the ASCII technology file associated with the layer-purpose pair.

For more information about the `techDisplays` subsection of the technology file, see techDisplays in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_layerPurposeID	The database identifier of the layer-purpose pair.

Value Returned

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`.

techGetLPPriorityInContext

```
techGetLPPriorityInContext
     (
     d_techID
     d_techLPId
     )
     => x_priority / nil
```

Description

Gets the drawing priority of the specified layer-purpose pair in the effective technology database rooted at the specified technology database.

The topmost layer-purpose pair (the layer-purpose pair with the highest number) is drawn on top. Each window, when drawn, references one particular incremental technology database graph. For example, you have five windows. Each window contains a different cellview and each cellview refers to a different incremental technology database graph. In this case, you would have five sets of “layer-purpose pair priorities” that are completely independent of each other. Each window would have priorities ranging from 0 to however many layer-purpose pairs exist in the corresponding effective technology database.

Arguments

d_techID	The database identifier of the technology database at the top of the graph.
l_techLPId	The database identifier of the layer-purpose pair.

Value Returned

x_priority	The drawing priority set on the layer-purpose pair in the specified technology database.
nil	The drawing priority was not set or the technology database or layer-purpose pair does not exist.

Example

```
techGetLPPriorityInContext(tfID lpId)
=> 3
```

Gets the drawing priority, `3`, of the layer-purpose pair identified by `lpId` in the technology database identified by `tfID`.

techGetLPProp

```
techGetLPProp(
     d_lpID
     t_propName
     )
     => g_propValue / nil
```

Description

Returns the value of the specified property from the specified LP. The `techLayerProperties` subsection in the `layerDefinitions` section of the ASCII technology file specifies special properties that you want to place on the layers in your design.

Arguments

d_lpID	The database identifier of the LP in the technology database.
t_propName	The name of the property.

Value Returned

g_propValue	The value of the property.
`nil`	The LP or property does not exist in the technology database or the property value is `nil`.

Example

```
techGetLPProp(lpID "myProp")
=> 0.1
```

Returns `0.1` as the value of the `myProp` property.

techGetLPsByPriority

```
techGetLPsByPriority(
     d_techFileID
     )
     => l_lps / nil
```

Description

Returns a list of database identifiers of all layer-purpose pairs (LPPs) from the specified technology file, from the lowest to the highest priority.

Arguments

d_techFileId	The ID of the technology file from which LPPs are to be extracted.

Value Returned

l_lps	A list of database identifiers of all LPPs from the specified technology file, ordered by priority.
nil	The technology file with the specified ID does not exist.

Example

```
techGetLPsByPriority( tech )
=> (db:0x1af0ce00 db:0x1bd40260 db:0x17a7b200 db:0x1b016520 db:0x1bd40390
    db:0x17aafca0 db:0x1bd404b0 db:0x17aa3b40 db:0x1b016400 db:0x1bd405d0
    …
   )
```

Returns a list of database identifiers of all LPPs from the specified technology file.

techGetOrderLeLsw

```
techGetOrderLeLsw(
     t_techLibName
     )
     => list( (t_layer t_purpose n_priority)...) / nil
```

Description

Returns the ordered list of layer-purpose pairs associated with the specified technology library as defined by the function `techOrderLeLsw`.

Arguments

t_techLibName	The technology file library name.

Value Returned

`list( (`*t_layer t_purpose n_priority*`)...)`

An ordered list of LPPs to display in the Palette assistant.

- *t_layer* is the layer name or number.
- *t_purpose* is the purpose name or number.
- *n_priority* is the priority in which the specified LPPs appear in the display order.

`nil` Unable to find the list of layer-purpose pairs.

Example

```
techGetOrderLeLsw("TechLib4")
techGetOrderLeLsw("TechLib4")
("Metal2" "drawing" 10.0)
("Via2" "drawing" 5.0)
\t )
```

Returns the ordered list of layer-purpose pairs associated with the technology library, `TechLib4`.

techGetOrderLPPriorityInContext

```
techGetOrderLPPriorityInContext(
     t_techLibName
     )
     => list( (t_layer t_purpose n_priority)...) / nil
```

Description

Returns the ordered list of layer-purpose pairs associated with the specified technology library as defined by the function `techOrderLPPriorityInContext`.

Arguments

t_techLibName	The technology file library name.

Value Returned

`list( (`*t_layer t_purpose n_priority*`)...)`

An ordered list of LPPs to display in the Palette assistant.

- *t_layer* is the layer name or number.
- *t_purpose* is the purpose name or number.
- *n_priority* is the priority in which the specified LPPs appear in the display order.

`nil` Unable to find the list of layer-purpose pairs.

Example

```
techGetOrderLPPriorityInContext("TechLib3")
```

Returns the ordered list of layer-purpose pairs associated with the technology library, `TechLib3`.

techGetOuterViaLayers

```
techGetOuterViaLayers(
     d_techID
     tx_viaLayer
     )
     => l_outerViaLayers / nil
```

Description

Given the via layer, or the middle layer of the via, returns the bottom and top layers used with the via layer in standard via definitions in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
tx_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

Value Returned

l_outerViaLayers	A list of the bottom and top layers used with the via layer in standard via definitions. The list has the following syntax: `( lt_bottom lt_top )` where, ■ *lt_bottom* is the bottom routing layer used with the specified via layer in standard via definitions. ■ *lt_top* is the top routing layer used with the specified via layer in standard via definitions.
nil	The technology database does not exist, or the specified layer is not used as a via layer in any standard via definitions.

Example

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

Returns the layers surrounding the specified via layer in standard via definitions in the technology database identified by `tfID`.

techGetPurposeAbbrev

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

Description

Returns the abbreviation of the specified purpose from the specified technology database. ASCII technology file location: `techPurposes` subsection in the `layerDefinitions` section; it lists the layer purposes used with technology data.

For more information about `techPurposes`, see techPurposes in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
tx_purpose	The purpose name or number. Valid values: The name or number of an existing purpose; a purpose number must be an integer, `1` through `128`, plus `256` through $2^{32}-1026$

Value Returned

t_purposeAbbrev	The abbreviation of the purpose name.
`nil`	The technology database does not exist, the purpose is not defined in the technology database, or the purpose has no abbreviation assigned.

Example

```
techGetPurposeAbbrev(tfID 257)
=> "drwg"
```

Returns the abbreviation of purpose number `257` defined in the `techPurposes` subsection of the `layerDefinitions` section of the technology database identified by `tfID`.

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

Returns the abbreviation of the `drawing` purpose defined in the `techPurposes` subsection of the `layerDefinitions` section of the technology database identified by `tfID`.

techGetPurposeName

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

Description

Returns the purpose name of the specified purpose number defined in the specified technology database. ASCII technology file location: `techPurposes` subsection in the `layerDefinitions` section; it lists the layer purposes used with technology data.

For more information about the `techPurposes` subsection of the technology file, see techPurposes in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
x_purposeNumber	The purpose number.

Value Returned

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

Example

```
techGetPurposeName(tfID 257)
=> "drawing"
```

Returns the purpose `drawing` for purpose number `257` in the technology database identified by `tfID`.

techGetPurposeNum

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

Description

Returns the purpose number associated with the specified purpose name from the specified technology database. ASCII technology file location: `techPurposes` subsection in the `layerDefinitions` section; it lists the layer purposes used with technology data.

For more information about `techPurposes`, see techPurposes in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
t_purposeName	The purpose name.

Value Returned

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

Example

```
techGetPurposeNum(tfID "drawing")
=> 257
```

Returns the purpose number `257` for the purpose `drawing` from the technology database identified by `tfID`.

techGetStampLabelLayers

```
techGetStampLabelLayers(
     d_techID
     )
     => l_labelLayers / nil
```

Description

Returns a list of the stamp label layers defined in the specified technology database. ASCII technology file location: `stampLabels` subsection in the `layerRules` section.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_labelLayers	A list of the label layers.
`nil`	The technology database does not exist or no `stampLabels` subsection is specified in the technology database.

techGetTrimLayerPairs

```
techGetTrimLayerPairs(
     d_techFileID
     tx_layer
     )
     => l_trimLayerPairs / nil
```

Description

Get list of layer pairs for given layer of `trim` material type.

Arguments

d_techFileID	The database identifier of the technology database.
tx_layer	The layer with material type `trim`.

Value Returned

l_trimLayerPairs	A list of layer pairs for given layer of `Trim` material type.
`nil`	The technology database does not exist or the layer is not defined in the technology database.

Example

```
techGetTrimLayerPairs( tx "via1" )
```

techGetTwoLayerProp

```
techGetTwoLayerProp(
     d_techID
     tx_layer1
     tx_layer2
     t_propName
     )
     => g_propValue / nil
```

Description

Returns the value of the specified two-layer property from the specified technology database. ASCII technology file location: `techLayerProperties` subsection in the `layerDefinitions` section; it specifies special properties that you want to place on the layers in your design.

For more information about `techLayerProperties`, see techLayerProperties in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
tx_layer1	The first layer. Valid values: The layer name or number or a list containing the layer name and purpose.
tx_layer2	The second layer. Valid values: The layer name or number or a list containing the layer name and purpose.
t_propName	The name of the property.

Value Returned

g_propValue	The value of the property.
`nil`	The technology database, layer(s), or property does not exist or the property value is `nil`.

Example

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

Returns `router`, the value of the `myProp` property set on the `via` and `metal1` layers in the technology database identified by `tfID`.

techGetViaLayers

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

Description

Returns a list of the sets of layers used in the standard via definitions defined in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_viaLayers	The list of sets of layers used in standard via definitions. The list has the following syntax: `( ( lt_bottom lt_via lt_top ) … )` where, ■ *lt_bottom* is the bottom routing layer of a set of layers used in standard vias. ■ *lt_via* is the middle layer of a set of layers used in standard vias, commonly called the via layer. ■ *lt_top* is the top routing layer of a set of layers used in standard vias.
nil	The technology database does not exist or does contain any standard via definitions.

Example

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

Lists the sets of layers used in standard via definitions in the technology database identified by `tfID`.

techHasLayerAnalysisAttribute

```
techHasLayerAnalysisAttribute(
     d_techfileID
     tx_layer
     t_attributeName
     )
     => t / nil
```

Description

Checks if a specified attribute has been set for a layer with a layer function that supports attribute analysis.

Arguments

d_techfileID	The database identifier of the technology database.
tx_layer	The layer name or number.
t_attributeName	The name of the attribute. The valid attribute names are: ■ `materialName`: The name of the material. ■ `thickness`: The thickness of the layer in microns. ■ `conductivity`: The conductivity of the layer in Siemens/meter. ■ `permittivity`: The dielectric constant of the layer. ■ `lossTangent`: The loss tangent of the layer.

Value Returned

`t`	The specified attribute has a value for the layer.
`nil`	The specified attribute does not have a value for the layer, or the layer does not have a function that supports layer analysis attributes or does not exist in the technology database.

Example

```
techHasLayerAnalysisAttribute(tech "Metal1" "lossTangent")
=> t
```

Returns `t` to indicate that a `lossTangent` attribute value has been set for the `Metal1` layer.

techIsLPValidBase

```
techIsLPValidBase(
     d_lpId
     )
     => t / nil
```

Description

Returns boolean indication whether the layer-purpose pair valid attribute is true.

Arguments

d_lpId	The database identifier of the layer-purpose pair.

Value Returned

t	The layer-purpose pair has a valid attribute.
nil	The layer-purpose pair does not have a valid attribute.

Example

```
techIsLPValidBase(d_lpID)
=> t
```

Returns `t` as the layer-purpose pair has a valid attribute.

techIsViaLayer

```
techIsViaLayer(
     d_techID
     tx_viaLayer
     )
     => t / nil
```

Description

Indicates whether the specified layer is defined as the middle, or via, layer in one or more standard via definitions in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
tx_viaLayer	The layer you want to check. Valid values: The layer name, the layer number, a list containing the layer name and layer purpose

Value Returned

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

Example

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

The `via` layer is the middle layer of one or more standard via definitions in the technology database identified by `tfID`.

techOrderLPPriorityInContext

```
techOrderLPPriorityInContext(
     tx_techfileLibName
     l_orderLpps
     )
     => t / nil
```

Description

Defines the relative priority for a dynamic set of layer-purpose pairs (LPPs) to be drawn on the canvas. The larger the number, the higher the priority and the lower the LPP is in the list. The LPP with the highest priority is drawn on top of all other LPPs.

This function is most useful when multiple technology databases are arranged in a technology graph. The LPPs specified are expected to exist in the graph rooted at the specified technology database. The function can be run on any variant of the process stack. It silently ignores any layers that are not present. Technology files on disk are not changed.

Arguments

tx_techfileLibName

The technology file library name.

l_orderLpps

An ordered list of layer-purpose pairs to be drawn on the canvas. The list has the following syntax:

`list ( list(` *tx_layer tx_purpose n_priority*`) … )`

where,

- *tx_layer* is the layer name or number.
- *tx_purpose* is the purpose name or number.
- *n_priority* is the priority in which the specified LPPs are drawn on the canvas.

Value Returned

`t`	The priority of LPPs was updated as specified.
`nil`	The technology database does not exist.

Example

Assume that the total LPPs in the effective technology database of topTech is 250 and that these LPPs are at the following positions:

- `"Metal1" "drawing"` is at 40.
- `"Metal2" "drawing"` is at 50.
- `"Metal3" "drawing"` is at 60.

The function is called as follows:

```
techOrderLPPriorityInContext("topTech"
    list(
        list("Metal1" "drawing" 1)
        list("Metal2" "drawing" 2)
        list("Metal3" "drawing" 3)
        )
)
```

The positions are updated as follows:

- `"Metal3" "drawing"` is at the last position.
- `"Metal2" "drawing"` is at the second position from below.
- `"Metal1" "drawing"` is at the third position from below.

techSetEquivLayer

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

Description

Appends the specified set of equivalent layers to the specified technology database. ASCII technology file location: `equivalentLayers` subsection in the `layerRules` section; it lists layers that represent the same kind of material.

For more information about the `equivalentLayers` subsection of the technology file, see equivalentLayers in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
l_equivLayers	A list of equivalent layers. The list has the following syntax: `list ( tx_layer … )` where, *tx_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

Value Returned

`t`	The equivalent layers were appended to the `equivalentLayers` subsection of the technology database, or the specified layers were already listed as equivalent layers in the `equivalentLayers` subsection.
`nil`	The technology database does not exist or one or more specified layers are not defined in the technology database.

Example

```
techSetEquivLayer(tfID list("metal1" "metal2"))
```

Appends the equivalent layer set of `metal1` and `metal2` to the `equivalentLayers` subsection of the technology database identified by `tfID`.

techSetEquivLayers

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

Description

Updates the `equivalentLayers` subsection of the specified technology database with the specified set or sets of equivalent layers. ASCII technology file location: `equivalentLayers` subsection in the `layerRules` section; it lists layers that represent the same kind of material. If an `equivalentLayers` subsection does not exist, this function creates one with the specified data. If the technology database already defines equivalent layers, this function deletes and replaces them with the specified data.

For more information about the `equivalentLayers` subsection of the technology file, see equivalentLayers in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
l_equivLayers	A list of lists indicating the equivalent layers to create. The list has the following syntax: `list ( list ( tx_layer … ) … )` where, *tx_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

Value Returned

`t`	`equivalentLayers` were created or re-created in the specified technology database.
`nil`	The technology database does not exist.

Example

```
techSetEquivLayers(tfID list(
list("metal1" "metal2")
list("via1" "via2" "via3")
list(list("metal3" "pin") "pinMetal")
)
```

Re-creates the `equivalentLayers` subsection of the technology database identified by `tfID` to define the specified equivalent layers

techSetLabelLayer

```
techSetLabelLayer(
     d_techID
     l_labelLayers
     )
     => t / nil
```

Description

Appends the specified label layer to the label layers list in the specified technology database. ASCII technology file location: `labels` subsection in the `layerRules` section; it lists label layers followed by their associated sets of layers of any type. If the label layer already has a specification in the `labels` subsection, this function replaces it with the new specification. If the `labels` subsection does not exist, this function creates one with the specified data.

Arguments

d_techID	The database identifier of the technology database.
l_labelLayers	A list of lists specifying label layers. The list has the format: `list ( tx_labelLayer1 tx_conLayer1 [ tx_conLayer2 … ] )` where, ■ *tx_labelLayer* is the label layer. ■ *tx_conLayer* is a layer of any type to be associated with the label layer.

Value Returned

`t`	The `labels` subsection has be successfully written with the specified data.
`nil`	The technology database does not exist.

techSetLabelLayers

```
techSetLabelLayers(
     d_techID
     l_labelLayers
     )
     => t / nil
```

Description

Replaces the label layers in the specified technology database. ASCII technology file location: `labels` subsection in the `layerRules` section; it lists label layers followed by their associated sets of layers of any type. If the `labels` subsection does not exist, this function creates one with the specified data. If the `labels` subsection does exist, this function deletes and replaces it with the specified data.

Arguments

d_techID	The database identifier of the technology database.
l_labelLayers	A list of lists specifying label layers. The list has the following syntax: `list ( list ( tx_labelLayer1 tx_conLayer1 [ tx_conLayer2 … ] ) … )` where, ■ *tx_labelLayer* is the label layer. ■ *tx_conLayer* is a layer of any type to be associated with the label layer.

Value Returned

`t`	The `labels` subsection has be successfully written with the specified data.
`nil`	The technology database does not exist.

techSetLayerAbbrev

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

Description

Updates the abbreviation of the specified layer in the specified technology database. ASCII technology file location: `techLayers` subsection in the `layerDefinitions` section; it lists the layers that can be used.

Note: When specifying a layer name abbreviation, consider the possibility of shortened names being displayed in selection windows.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The layer name or number.
t_layerAbbrev	The new abbreviation for the layer name. Valid values: Any string of seven characters or less

Value Returned

`t`	The layer abbreviation was successfully updated.
`nil`	The technology database does not exist or does not define the layer.

Example

```
techSetLayerAbbrev(tfID 16 "met2")
```

Updates the abbreviation of layer number `16` to `met2` in the technology database identified by `tfID`.

```
techSetLayerAbbrev(tfID "metal2" "met2")
```

Updates the abbreviation of the `metal2` layer to `met2` in the technology database identified by `tfID`.

techSetLayerAnalysisAttribute

```
techSetLayerAnalysisAttribute(
     d_techfileID
     tx_layer
     t_attributeName
     g_attributeValue
     )
     => t / nil
```

Description

Sets a specified attribute value for a layer with a layer function that supports attribute analysis.

FArguments

d_techfileID	The database identifier of the technology database.
tx_layer	The layer name or number.
t_attributeName	The name of the attribute. The valid attribute names are: ■ `materialName`: The name of the material. ■ `thickness`: The thickness of the layer in microns. ■ `conductivity`: The conductivity of the layer in siemens/meter. ■ `permittivity`: The dielectric constant of the layer. ■ `lossTangent`: The loss tangent of the layer.
g_attributeValue	The value of the attribute. It can be String or Double.

Value Returned

`t`	The specified attribute has been set for the layer.
`nil`	The attribute value was not set. The layer does not have a function that supports layer analysis attributes, or it does not exist in the technology database.

Example

```
techSetLayerAnalysisAttribute(tech "Metal1" "materialName" "Copper")
=> 0.9
```

Sets `Copper` as the value of the `materialName` attribute for the `Metal1` layer.

techSetLayerAnalysisAttributeLock

```
techSetLayerAnalysisAttributeLock(
     d_techID
     s_analysisAttributeLockType
     g_value
     )
     => t / nil
```

Description

Sets a lock on layer analysis attributes for changes related to addition of new layers or updates to attributes.

The following analysis attribute locks are supported for this function:

- `lockAddLayers`: Prevents addition or removal of layers from the stack but their values can be changed.
- `lockValuesChange`: Prevents change of layer attributes. However, layers can be added or removed.

Arguments

d_techID	The database identifier of the technology database.
s_analysisAttributeLockType	The type of analysis attribute lock. Valid values: `lockAddLayers`, `lockValuesChange`.
g_value	Enables or disables the lock. Valid values: `t`, `nil`.

Value Returned

`t`	Lock on layers analysis attributes has been successfully set.
`nil`	Lock on layers analysis attributes has not been set.

Example

```
techID = techGetTechFile(geGetEditCellView())
lockType = "lockAddLayers"
when(techSetLayerAnalysisAttributeLock(techID lockType t)
    print("'Add layers' lock activated")
)
```

techSetLayerFunction

```
techSetLayerFunction(
     d_techID
     tx_layer
     g_function
     )
     => t / nil
```

Description

When the specified layer is not already assigned a function (material), sets the function assignment in the specified technology database. If the layer is already assigned a function, `techSetLayerFunction` returns a message to that effect and does not update the technology database. ASCII technology file location: `functions` subsection in the `layerRules` section; it assigns functions to layers. If a `functions` subsection does not exist, this function creates one with the specified data.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The layer name or layer number or a list containing the layer name and purpose.
g_function	The layer function.

Value Returned

`t`	The function was set.
`nil`	The technology database does not exist, the layer is not defined in the technology database, the requested function is not a valid function, or the layer already has another function assigned to it.

Example

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

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

```
techSetLayerFunction(tfID "Nwell" "pwell")
=> *WARNING* techSetLayerFunction: layer "Nwell" already has function: nwell
nil
```

Does not make the requested function assignment because the layer is already assigned a function.

```
techSetLayerFunction(tfID "Metal1" "unknown")
=> *WARNING* techSetLayerFunction: Illegal function specified "unknown"
nil
```

Does not make the requested function assignment because the specified function is not a valid function.

techSetLayerFunctions

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

Description

When the specified layers are not already assigned functions (materials), sets the function assignments in the specified technology database. If any layer is already assigned a function, `techSetLayerFunctions` returns a message to that effect and does not change that function assignment, but does change the others. ASCII technology file location: `functions` subsection in the `layerRules` section; it assigns functions to layers. If a `functions` subsection does not exist, this function creates one with the specified data.

For more information about layer functions, see functions in Virtuoso *Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
l_layerFunctionsList	A list of lists specifying the layer names and functions to assign to them. The list has the following syntax: `list( list( tx_layer g_function ) … )` where, ■ *tx_layer* is the layer name or number or a list containing the layer name and purpose. ■ *g_function* is the layer function.

Value Returned

`t`	The functions are assigned to the layers. **Note:** If some functions cannot be set for any reason, the software returns error messages for those, sets the ones that can be set, and returns `t`.
`nil`	The technology database does not exist, the layers are not defined in the technology database, the functions are not valid functions, or all of the layers already have other functions assigned to them.

Example

```
techSetLayerFunctions(tfID list(list("poly2" "poly") list("metal3" "metal")))
=> t
```

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

techSetLayerMaskNumber

```
techSetLayerMaskNumber(
     d_techID
     tx_layer
     x_maskNumber
     )
     => t / nil
```

Description

Updates the mask number of the specified layer in the specified technology database. ASCII technology file location: `functions` subsection in the `layerRules` section; it assigns functions and mask numbers to layers.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The layer name or number or a list containing the layer name and purpose.
x_maskNumber	The mask number for the layer. Valid values: Any integer

Value Returned

`t`	The mask number is assigned to the specified layer.
`nil`	The technology database does not exist or there is no `functions` definition for the specified layer.

Example

```
techSetLayerMaskNumber(tfID "metal1" 1)
=> t
```

Assigns the mask number, `1`, to the layer `metal1` in the technology database identified by `tfID`.

echSetLayerMfgResolution

```
techSetLayerMfgResolution(
     d_techID
     tx_layer
     g_value
     )
     => t / nil
```

Description

Sets or updates the layer manufacturing grid resolution to be applied to the specified layer in the specified technology specified technology database. ASCII technology file location: `mfgResolutions` subsection in the `layerRules` section. If a `mfgResolutions` subsection does not exist, this function creates one with the specified data.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The layer name or layer number or a list containing the layer name and layer purpose. **Note:** If you specify a layer and purpose, this function sets the layer manufacturing resolution on all occurrences of the layer, no matter what purposes may be paired with it.
g_value	The manufacturing grid resolution to apply to the layer.

Value Returned

`t`	The manufacturing grid resolution is assigned to be applied to the layer.
`nil`	The technology database does not exist or the layer is not defined in the technology database.

Example

```
techSetLayerMfgResolution(tfID "metal1" 0.0020)
=> t
```

Assigns a manufacturing grid resolution of `0.0020` to be applied to the layer `metal1`.

techSetLayerMfgResolutions

```
techSetLayerMfgResolutions(
     d_techID
     l_layers
     )
     => t / nil
```

Description

Replaces the layer manufacturing grid resolution data in the specified technology database. ASCII technology file location: `mfgResolutions` subsection in the `layerRules` section. If a `mfgResolutions` subsection does not exist, this function creates one with the specified data.

Arguments

d_techID	The database identifier of the technology database.
l_layers	A list of lists containing the layer and manufacturing grid resolution data. The list has the following syntax: `list ( list ( tx_layer g_value ) … )` where, ■ *tx_layer* is the layer name or layer number. ■ *g_value* is the manufacturing grid resolution to apply to the layer.

Value Returned

`t`	The manufacturing grid resolutions are assigned to be applied to the layers.
`nil`	The technology database does not exist or one or more layers are not defined in the technology database.

Example

```
techSetLayerMfgResolutions(tfID list(list ("metal1" 0.0020) list("metal2" 0.0030))
=> t
```

Assigns a manufacturing grid resolution of `0.0020` to be applied to the layer `metal1` and `0.0030` to `metal2`.

techSetLayerName

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

Description

Updates the name of the specified layer in the specified technology database. ASCII technology file location: `techLayers` subsection in the `layerDefinitions` section; it lists the layers that can be used.

Note: When specifying a layer name, consider the possibility of shortened names being displayed in selection windows.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The current layer name or the layer number.
t_layerName	The new layer name. Valid values: Any string

Value Returned

`t`	The layer name was successfully updated.
`nil`	The technology database does not exist or does not define the layer.

Example

```
techSetLayerName(tfID 16 "metal2")
```

Updates the name of layer number `16` to `metal2` in the technology database identified by `tfID`.

```
techSetLayerName(tfID "Metal2" "metal2")
```

Updates the name of layer `Metal2` to `metal2` in the technology database identified by `tfID`.

techSetLayerProp

```
techSetLayerProp(
     d_techID
     tx_layer
     l_propertyValue
     )
     => t / nil
```

Description

Updates the value of the specified layer property in the specified technology database. ASCII technology file location: `techLayerProperties` subsection in the `layerDefinitions` section; it specifies special properties that you want to place on the layers in your design. If the `techLayerProperties` subsection 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 `techLayerProperties`, see techLayerProperties in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The layer name or layer number or 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 )` where, ■ *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

Value Returned

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

Example

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

Sets the `myCorpCADControlValue` property to `t` on the `metal1` layer in the technology database identified by `tfID`.

techSetLayerRoutingGrid

```
techSetLayerRoutingGrid(
     d_techID
     tx_layer
     t_preferredDir
     [ g_pitch [ g_offset ] ]
     )
     => t / nil
```

Description

Updates the routing direction, pitch, and offset for the specified layer in the specified technology database. The pitch is the minimum allowable spacing, center-to-center, between two regular geometries on different nets. The offset is the distance between the placement grid and the routing grid when there is a routing grid between two placement grids.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The name or number of the layer to which to apply the routing definition.
t_preferredDir	The keyword used to indicate how you want the place-and-route software to use the layer. Valid values: `none`, `horizontal`, `vertical`, `leftDiag`, `rightDiag`
g_pitch	The pitch, in user units, for the routing grid of the layer.
g_offset	The offset, in user units, for the routing grid of the layer.

Value Returned

`t`	The specified routing direction, pitch, and offset are set the specified layer in the specified technology database.
`nil`	The technology database does not exist or the specified layer is not defined.

Example

```
techSetLayerRoutingGrid(tfID "METAL1" "horizontal")
=> t
```

Sets the layer routing definition on `METAL1` to `horizontal` in the technology database identified by `tfID`.

techSetLayerRoutingGrids

```
techSetLayerRoutingGrids(
     d_techID
     l_layerRoutingGrids
     )
     => t / nil
```

Description

Replaces the layer routing direction, pitch, and offset for the specified layers in the specified technology database. The pitch is the minimum allowable spacing, center-to-center, between two regular geometries on different nets. The offset is the distance between the placement grid and the routing grid when there is a routing grid between two placement grids.

Arguments

d_techID The database identifier of the technology database.

l_layerRoutingGrids

A list of layers and their routing specifications. The list has the following syntax:

```
list ( list ( tx_layer t_preferredDir
[ g_pitch [ g_offset ] ] ) … )
```

where,

- *tx_layer* is the name or number of the layer to which to apply the routing definition.
- *t_preferredDir* is the keyword used to indicate how you want the place-and-route software to use the layer.

  Valid values: `none`, `horizontal`, `vertical`, `leftDiag`, `rightDiag`
- *g_pitch* is the pitch, in user units, for the routing grid of the layer.
- *g_offset* is the offset, in user units, for the routing grid of the layer.

Value Returned

`t`	The specified routing direction, pitch, and offset are set the specified layer in the specified technology database.
`nil`	The technology database does not exist or one or more of the specified layers are not defined.

Example

```
techSetLayerRoutingGrids(tfID '(("Metal2" "horizontal" 0.31)
("Metal3" "vertical" 0.51 0.05)))
=> t
```

On `METAL2`, sets the routing direction to `horizontal` and the pitch to `0.31`, and on `METAL3`, sets the routing direction to `vertical`, the pitch to `0.51`, and the offset to `0.05` in the technology database identified by `tfID`.

techSetLPAttr

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

Description

Updates layer attributes for the specified layer-purpose pair in the current technology database. The layer attributes are specified as a list and include the display priority, visibility, selectability, if the layer is visible when dragged, and validity. ASCII technology file location: `techDisplays` and `techLayerPurposePriorities` subsections of the `layerDefinitions` section.

For information about the `techLayerPurposePriorities` subsection of the technology file, see techLayerPurposePriorities in *Virtuoso Technology Data ASCII Files Reference*. For information about the `techDisplays` subsection of the technology file, see techDisplays in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_layerPurposeID	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 ] )
```

where,

- *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.

  Valid values: `t`, `nil`

- *g_dragEnable* indicates whether you can drag a shape created with the layer-purpose pair with the Virtuoso® Layout Suite L 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`

Value Returned

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

Example

```
techSetLPAttr(lpID list( 4 t t t t t))
=> t
```

Sets attributes for the layer-purpose pair identified by `lpID` in the current technology database.

techSetLPPacketName

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

Description

Updates the display packet assigned to the specified layer-purpose pair in the current technology database. ASCII technology file location: `techDisplays` subsection. The display packet must be defined in the display resource file before you can assign it.

For more information about the `techDisplays` subsection of the technology file, see techDisplays in *Virtuoso Technology Data ASCII Files Reference*. For more information about defining display packets in the display resource file, see *Virtuoso Technology Data ASCII Files Reference*.

Arguments

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

Value Returned

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

Example

```
techSetLPPacketName(lpID "redsolid_S")
=> t
```

Assigns the display packet `redsolid_S` to the layer-purpose pair identified by `lpID` in the current technology database.

techSetLPPriorityInContext

```
techSetLPPriorityInContext(
     d_tfId
     d_techLPId
     x_priority
     )
     => d_lppId / nil
```

Description

Sets the drawing priority of a layer-purpose pair in an effective technology database. The larger the number, the higher the priority. The layer-purpose pair must exist in the graph rooted at the specified technology database; it does not need to exist locally in the specified technology database.

The following three cases apply:

- If *d_techLpId* exists in *d_tfId*, the priority is reset and the function returns *d_techLpId*.
- If *d_techLPId* is not defined locally in *d_tfId* and no equivalent layer-purpose pair exists in *d_tfId*, a clone of *d_techLPId* is created in the technology database *d_tfID* and the function returns this newly created layer-purpose pair.
- If *d_techLpId* does not exist in *d_tfId*, but an equivalent layer-purpose pair exists in *d_tfId*, the function returns `nil`.

Note: The layer-purpose pair with the highest priority is drawn on top of all other layer-purpose pairs.

Arguments

d_tfId	The database identifier of the technology database at the top of the graph.
l_techLPId	The database identifier of the layer-purpose pair.
x_priority	The drawing priority to be set on the layer-purpose pair.

Value Returned

d_lppId	The newly created layer-purpose pair or the layer-purpose pair with ID *l_techLPId* if it is already locally defined in the technology database with ID *d_tfId*.
nil	The drawing priority was not set; the technology database or layer-purpose pair does not exist or does not satisfy the required criteria; or *d_techLpId* does not exist in *d_tfId*, but an equivalent layer-purpose pair exists in *d_tfId*.

Example

```
techSetLPPriorityInContext(tfID lpId 3)
```

Sets the drawing priority of the layer-purpose pair identified by `lpId` to `3` in the effective technology database rooted at the technology database identified by `tfID`.

techSetStampLabelLayer

```
techSetStampLabelLayer(
     d_techID
     l_labelLayers
     )
     => t / nil
```

Description

Appends the specified stamp label layers to the label layers list in the specified technology database. ASCII technology file location: `stampLabels` subsection in the `layerRules` section; it lists label layers followed by their associated sets of layers of connected type. If the label layer already has a specification in the `stampLabels` subsection, this function replaces it with the new specification. If the `stampLabels` subsection does not exist, this function creates one with the specified data.

Arguments

d_techID	The database identifier of the technology database.
l_labelLayers	The database identifier of the technology database. A list of lists specifying label layers. The list has the following syntax: `list ( tx_labelLayer1 tx_conLayer1 [ tx_conLayer2 … ] )` where, ■ *tx_labelLayer* is the stamp label layer. ■ *tx_conLayer* is a layer of connected type to be associated with the stamp label layer.

Value Returned

`t`	The `stampLabels` subsection has be successfully written with the specified data.
`nil`	The technology database does not exist.

techSetStampLabelLayers

```
techSetStampLabelLayers(
     d_techID
     l_labelLayers
     )
     => t / nil
```

Description

Replaces the specified stamp label layers in the specified technology database. ASCII technology file location: `stampLabels` subsection in the `layerRules` section; it lists label layers followed by their associated sets of layers of connected type. If the `stampLabels` subsection does not exist, this function creates one with the specified data. If the `stampLabels` subsection does exist, this function deletes and replaces it with the specified data.

Arguments

d_techID	The database identifier of the technology database.
l_stampLabelLayers	A list of lists specifying label layers. The list has the following syntax: `list ( list ( tx_labelLayer1 tx_conLayer1 [ tx_conLayer2 …] ) … )` where, ■ *tx_labelLayer* is the label layer. ■ *tx_conLayer* is a layer of connected type to be associated with the label layer.

Value Returned

`t`	The `stampLabels` subsection has be successfully written with the specified data.
`nil`	The technology database does not exist.

techSetTwoLayerProp

```
techSetTwoLayerProp(
     d_techID
     tx_layer1
     tx_layer2
     l_propertyValue
     )
     => t / nil
```

Description

Updates the value of the specified two-layer property in the specified technology database. ASCII technology file location: `techLayerProperties` subsection in the `layerDefinitions` section; it specifies special properties that you want to place on the layers in your design. If the `techLayerProperties` subsection 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 `techLayerProperties`, see techLayerProperties in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
tx_layer1	The first layer. Valid values: The layer name or layer number or a list containing the layer name and purpose
tx_layer2	The second layer. Valid values: The layer name or layer number or 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 )
```

where,

- *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

Value Returned

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

Example

```
techSetTwoLayerProp(tfID "metal1" "via2" list("myCorpCADControlValue" t))
=> t
```

Sets the two-layer property `myCorpCADControlValue` property to `t` on layers `metal1` and `via2` in the technology database identified by `tfID`.

techSupportsLayerAnalysisAttributes

```
techSupportsLayerAnalysisAttributes(
     d_techfileID
     tx_layer
     )
     => t / nil
```

Description

Checks if the given layer has a function that supports layer analysis attributes.

Functions that support layer analysis attributes are: `buriedN`, `buriedP`, `cut`, `deepNimplant`, `deepNwell`, `deepPimplant`, `deepPwell`, `dielectric`, `diestack`, `diff`, `li`, `metal`, `mimcap`, `ndiff`, `nimplant`, `nwell`, `padMetal`, `passivationCut`, `pdiff`, `pimplant`, `pipcap`, `poly`, `pwell`, `substrate`, `tsv`, and `tsvMetal`.

Arguments

d_techfileID	The database identifier of the technology database.
tx_layer	The layer name or number.

Value Returned

`t`	The layer has a function that supports layer analysis attributes.
`nil`	The layer does not have a function that supports layer analysis attributes, or it does not exist in the technology database.

Example

```
techSupportsLayerAnalysisAttributes(tech "Metal1")
=> t
```

Returns `t` to indicate that `Metal1` has a function that supports layer analysis attributes.

4

Physical and Electrical Constraints Functions

The physical and electrical constraints functions let you update, establish, and return technology database physical and electrical constraints and layer attributes—update or return constraints from the `foundry` constraint group and layer attributes from the `layerRules` section of the technology file.

Note: Adding new constraints to a technology database with SKILL functions sometimes results in the creation of a new section of an already existing category. For example, adding a new spacing rule results in the addition of a new `spacings()` section somewhere after the last `spacings()` section that already exists in the technology database. This is done to preserve the correct order of the constraints; as long as the constraints are in the proper order in the technology database, multiple sections of the same category can coexist without problems. The descriptions for SKILL functions that can create a new section indicate that fact with a statement such as, "If the specified constraint does not already exist, this function creates a new `spacings` section containing the constraint."

The functions are categorized as:

- Physical constraints functions

  These functions set and get spacing rules and via stack limits. This category of functions are listed below:

  - `techCreateSpacingRuleTable`
  - `techGetOrderedSpacingRule`
  - `techGetOrderedSpacingRules`
  - `techGetSpacingRule`
  - `techGetSpacingRules`
  - `techGetSpacingRuleTable`
  - `techGetSpacingRuleTableEntry`

- techGetSpacingRuleTables
- techGetViaStackLimit
- techGetViaStackLimits
- techSetOrderedSpacingRule
- techSetSpacingRule
- techSetSpacingRuleTableEntry
- techSetViaStackLimit
- techSetViaStackLimits

■ Electrical constraints and layer attributes constraints

These functions set and get electrical constraints. This category of functions are listed below:

- techCreateElectricalRuleTable
- techGetCurrentDensityRules
- techGetCurrentDensityRuleTable
- techGetCurrentDensityRuleTables
- techGetElectricalRule
- techGetElectricalRules
- techGetElectricalRuleTable
- techGetElectricalRuleTableEntry
- techGetElectricalRuleTables
- techGetOrderedElectricalRule
- techGetOrderedElectricalRules
- techSetElectricalRule
- techSetElectricalRuleTableEntry
- techSetOrderedElectricalRule

Physical Constraints Functions

The physical constraints functions set and get spacing rules and via stack limits.

techCreateSpacingRuleTable

```
techCreateSpacingRuleTable(
     d_techID
     t_constraintName
     l_indexDefinitions
     tx_layer1
     [ tx_layer2 ]
     )
     => t / nil
```

Description

Creates a spacing table and indexes for the named constraint in the specified technology database. This new table does not overwrite an existing one. ASCII technology file location: `spacingTables` subsection in the `foundry` constraint group. To create table entries, use techSetSpacingRuleTableEntry. If a `spacingTables` subsection does not exist, this function creates one.

Arguments

d_techID	The database identifier of the technology database.
t_constraintName	The name of the spacing constraint.
l_indexDefinitions	A list defining the name of the index or indexes for the table. The list has the following syntax: `list( [ t_index1Name ] nil nil [ t_index2Name ] nil nil )` where, ■ *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 ■ *t_index2Name* is the name of the second dimension of a two-dimensional table. Valid values: Any number or string

tx_layer1	The first layer on which to apply the table constraint. Valid values: The layer name, the layer number
tx_layer2	The optional second layer on which to apply the table constraint. Valid values: The layer name, the layer number

Value Returned

`t`	The spacing table was created.
`nil`	The spacing table was not created.

Examples

```
techCreateSpacingRuleTable(tfID "minSpacing"
list("width" nil nil "length" nil nil) "Metal1")
=> t
```

Creates a 2D `minSpacing` table with indexes `width` and `length` for `Metal1` in the `foundry` constraint group:

```
spacingTables(
;( constraint layer1 [layer2]
;   (( index1Definitions    [index2Defintions]) [defaultValue] )
;   ( table) )
;( ----------------------------------------)
 ( minSpacing                  "Metal1"
    (("width"  nil  nil     "length"  nil  nil))
     (
      )
 )
) ;spacingTables
```

`defaultValue` is the value that is currently not obeyed by any tool. Therefore, if you query for a value that is not present in the table, `0` is returned irrespective of the value specified in this field.

```
( minSpacing                  "Metal1"
    (("width"  nil  nil     "length"  nil  nil)0.08)
     (
      (0.2 0.38)   0.11
      (0.2 0.42)   0.11
      (0.2 1.5)    0.11
      )
 )
```

Returns 0 if you query (0.1 0.1) from the table. This is because none of the entries in the table fits the (0.1 0.1) criterion. The lowest entry is width, which is greater than 0.2 and parallel run length is greater than 0.38. So, all queries below this particular value will return 0.

It is recommended that you provide a full table for greater clarity, as shown below:

```
( minSpacing                  "Metal1"
    (("width"  nil  nil     "length"  nil  nil)0.08)
     (

      (0.0 0.0)    0.09
      (0.0 0.38)   0.09
      (0.0 0.42)   0.09
      (0.0 1.5)    0.09
      (0.2 0.0)    0.09
      (0.2 0.38)   0.11
      (0.2 0.42)   0.11
      (0.2 1.5)    0.11
      )
 )
```

If you now query (0.1 0.1), you will get 0.09.

techGetOrderedSpacingRules

```
techGetOrderedSpacingRules(
     d_techID
     )
     => l_spacingRules / nil
```

Description

Returns a list of all of the ordered spacing constraints defined in the specified technology database. ASCII technology file location: `orderedSpacings` subsection(s) in the `foundry` constraint group.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_spacingRules	A list of lists containing the ordered spacing constraint settings. The list has the following syntax: `( ( t_constraint g_value lt_layer1 lt_layer2 ) … )` where, ■ *t_constraint* is the name of the spacing constraint. ■ *g_value* is the value of the spacing constraint. ■ *lt_layer1* is the first layer. ■ *lt_layer2* is the second layer.
`nil`	The technology database does not exist or there are no spacing constraints defined.

Example

```
techGetOrderedSpacingRules(tfID)
=> (("minEnclosure" 0.005 "Metal1" "Via1")
       ("minEnclosure" 0.005 "Metal2" "Via1")
       ("minEnclosure" 0.005 "Metal2" "Via2")
       ("minEnclosure" 0.005 "Metal3" "Via2")
       ("minEnclosure" 0.005 "Metal3" "Via3")
```

```
        ("minEnclosure" 0.005 "Metal4" "Via3")
        ("minEnclosure" 0.005 "Metal4" "Via4")
        ("minEnclosure" 0.005 "Metal5" "Via4")
        ("minEnclosure" 0.005 "Metal5" "Via5")
        ("minEnclosure" 0.005 "Metal6" "Via5")
        ("minEnclosure" 0.005 "Metal6" "Via6")
        ("minEnclosure" 0.005 "Metal7" "Via6")
        ("minEnclosure" 0.03  "Metal7" "Via7")
        ("minEnclosure" 0.05  "Metal8" "Via7")
        ("minEnclosure" 0.03  "Metal8" "Via8")
        ("minEnclosure" 0.05  "Metal9" "Via8")
        ("minEnclosure" 0.0   "Metal1" "Cont")
        ("minEnclosure" 0.2   "Oxide" "Poly")
        ("minEnclosure" 0.14  "Nimp" "Oxide")
        ("minEnclosure" 0.18  "Nimp" "Poly")
        ("minEnclosure" 0.12  "Pimp" "Poly")
        ("minEnclosure" 0.18  "Pimp" "Oxide")
        ("minEnclosure" 0.14  "Nwell" "Oxide")
        ("minEnclosure" 10.0  "Metal1" "Metal2")
    )
```

Returns the ordered spacing constraints defined in the `foundry` constraint group of the technology database identified by `tfID`.

techGetSpacingRules

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

Description

Returns a list of the spacing constraints defined in the specified technology database. ASCII technology file location: `spacings` subsection(s) in the `foundry` constraint group.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_spacingRules	A list of lists containing the spacing constraint settings. The list has the following syntax: `( ( t_constraint` *g_value t_layer1* `[` *t_layer2* `] ) … )` where, ■ *t_constraint* is the name of the spacing constraint. ■ *g_value* is the value of the spacing constraint. ■ *t_layer1* is the first layer. ■ *t_layer2* is the second layer. Returned only for two-layer spacing constraints.
`nil`	The technology database does not exist or there are no spacing constraints defined.

Example

```
techGetSpacingRules(tfID)
=> (("minWidth" 2
    ("Metal1" "slot")
)
("minSameNetSpacing" 0.12
    ("Metal1" "slot")
```

```
)
("minSpacing" 0.12
    ("Metal1" "slot")
("minWidth" 2
    ("Metal2" "slot")
)
("minSameNetSpacing" 0.14
    ("Metal2" "slot")
)
("minSpacing" 0.14
    ("Metal2" "slot")
)

…
("minSameNetSpacing" 0.12 "Metal1")
("minSpacing" 0.12 "Metal1")
("minWidth" 0.14 "Via1")
("minSameNetSpacing" 0.15 "Via1")
("minSameNetSpacing" 0.14 "Metal2")
("minSpacing" 0.14 "Metal2")
("minWidth" 0.14 "Via2")
("minSameNetSpacing" 0.15 "Via2")

…
("stackable" t "Metal1" "Metal1")
("stackable" t "Metal2" "Metal2")
("stackable" t "Via1" "Via2")
("stackable" t "Via2" "Via3")
…
)
```

Returns the spacing constraints defined in the `foundry` constraint group of the technology database identified by `tfID`. (… indicates that return values have been deleted from the original sample listing for purposes of brevity.)

techGetSpacingRuleTable

```
techGetSpacingRuleTable(
     d_techID
     t_constraintName
     tx_layer1
     [ tx_layer2 ]
     )
     => l_tables / nil
```

Description

Returns the table data for the specified constraint defined in the specified technology database and applied to the specified layer or layers. ASCII technology file location: `spacingTables` subsection in the `foundry` constraint group.

Arguments

d_techID	The database identifier of the technology database.
t_constraintName	The name of the spacing constraint.
tx_layer1	The first layer on which to apply the table constraint. Valid values: The layer name, the layer number
tx_layer2	The optional second layer on which to apply the table constraint. Valid values: The layer name, the layer number

Value Returned

l_tables — A list defining the table entries. The list has the following syntax:

```
( ( l_indexDefinitions ) l_table )
```

where,

- *l_indexDefinitions* is a list naming the table indexes. This list has the following syntax:

  ```
  ( nt_index1Name  nil nil
  [ nt_index2Name  nil nil] )
  ```

  where, *nt_index1Name* is the name of the first dimension of a two-dimensional table or the only dimension of a one-dimensional table and *nt_index2Name* is the name the second dimension of a two-dimensional table.

- *l_table* is a list of the table entries.

nil — The technology database does not exist or there are no spacing tables defined for the specified constraint on the specified layer or layers.

Example

```
techGetSpacingRuleTable(tfID "minSpacing" "Metal1"
=> (("width" nil nil "length" nil nil
      )
      (0.0005 0.0005) 0.12
      (0.0005 0.5605) 0.18
      (0.0005 1.5005) 0.5
      (0.0005 3.0005) 0.9
      (0.0005 7.5005)
      2.5
      (0.1805 0.0005) 0.18
      (0.1805 0.5605) 0.18
      (0.1805 1.5005) 0.5
      (0.1805 3.0005) 0.9
      (0.1805 7.5005)
      2.5
      (1.5005 0.0005) 0.5
      (1.5005 0.5605) 0.5
      (1.5005 1.5005) 0.5
      (1.5005 3.0005) 0.9
      (1.5005 7.5005)
      2.5
      (3.0005 0.0005) 0.9
      (3.0005 0.5605) 0.9
      (3.0005 1.5005) 0.9
      (3.0005 3.0005) 0.9
      (3.0005 7.5005)
```

```
      2.5
      (4.5005 0.0005) 1.5
      (4.5005 0.5605) 1.5
      (4.5005 1.5005) 1.5
      (4.5005 3.0005) 1.5
      (4.5005 7.5005)
      2.5
      (7.5005 0.0005) 2.5
      (7.5005 0.5605) 2.5
      (7.5005 1.5005) 2.5
      (7.5005 3.0005) 2.5
      (7.5005 7.5005)
      2.5
)
```

Returns the table data for the `minSpacing` constraint applied to layer `Metal1` in the `foundry` constraint group of the technology database identified by `tfID`.

techGetSpacingRuleTableEntry

```
techGetSpacingRuleTableEntry(
     d_techID
     t_constraintName
     g_index | l_index
     tx_layer1
     [ tx_layer2 ]
     )
     => g_value / nil
```

Description

Returns the value of the table constraint in the specified technology database that (a) is in the `foundry` constraint group, (b) is applied to the specified layer or layers, and (c) is the value for the table entry with the specified index or indexes.

Note: After running the `cdb2oa` translator on a CDBA table that contains table values of `nil`, the resultant OA table will display the CDBA `nil` values as OA default values when called by `techGetSpacingRuleTableEntry()`.

Arguments

d_techID	The database identifier of the technology database.
t_constraintName	The name of the spacing table constraint.
g_index	The index in a one-dimensional table for which to return the table value.
l_index	A list specifying the pair of indexes in a two-dimensional table for which to return the table value. The list has the following syntax: `list( g_index1 g_index2 )` where, ■ *g_index1* is the first table index. ■ *g_index2* is the second table index.
tx_layer1	The first layer on which the table constraint is applied. Valid values: The layer name, the layer number

tx_layer2	The optional second layer on which the table constraint is applied. Valid values: The layer name, the layer number

Value Returned

g_value	The value of the specified index.
nil	The technology database does not exist or the spacing table constraint is not defined.

Example

```
techGetSpacingRuleTableEntry(tfID "minSpacing" 0 "via")
=> 5.5
```

Returns the value (`5.5`) of the index `0` from the one-dimensional `minSpacing` table applied to layer `via` in the technology database identified by `tfID`.

```
techGetSpacingRuleTableEntry(tfID "minSpacing" list(0.0005 0.0005) "Metal2")
=> 0.14
```

Returns the value (`0.14`) of the index (`0.0005 0.0005`) from the two-dimensional `minSpacing` table applied to layer `Metal1` in the technology database identified by `tfID`.

techGetSpacingRuleTables

```
techGetSpacingRuleTables(
     d_techID
     )
     => l_tables / nil
```

Description

Returns a list of the spacing tables and the layers to which they apply as defined in the specified technology database. ASCII technology file location: `spacingTables` subsection in the `foundry` constraint group.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_tables	A list of the table spacing constraint names and the layers to which they apply. The list has the following syntax: `( ( t_constraintName tx_layer1 [ tx_layer2 ] ) … )` where, ■ *t_constraintName* is the name of the spacing constraint. ■ *tx_layer1* is the first layer on which the table constraint is applied. ■ *tx_layer2* is the optional second layer on which the table constraint is applied.
`nil`	The technology database does not exist or there are no spacing constraints tables defined.

Example

```
techGetSpacingRuleTables(tfID)
=> (("minDensity" "Metal1")
       ("maxDensity" "Metal1")
       ("minDensity" "Metal2")
```

```
        ("maxDensity" "Metal2")
        ("minDensity" "Metal3")
        ("maxDensity" "Metal3")
        ("minDensity" "Metal4")
        ("maxDensity" "Metal4")
        ("minDensity" "Metal5")
        ("maxDensity" "Metal5")
        ("minDensity" "Metal6")
        ("maxDensity" "Metal6")
        ("minDensity" "Metal7")
        ("maxDensity" "Metal7")
        ("minDensity" "Metal8")
        ("maxDensity" "Metal8")
        ("minDensity" "Metal9")
        ("maxDensity" "Metal9")
        ("minNumCut" "Cont")
        ("minNumCut" "Via1")
        ("minNumCut" "Via2")
        ("minNumCut" "Via3")
        ("minNumCut" "Via4")
        ("minNumCut" "Via5")
        ("minNumCut" "Via6")
        ("minNumCut" "Via7")
        ("minNumCut" "Via8")
        ("minSpacing" "Metal1")
        ("minSpacing" "Metal2")
        ("minSpacing" "Metal3")
        ("minSpacing" "Metal4")
        ("minSpacing" "Metal5")
        ("minSpacing" "Metal6")
        ("minSpacing" "Metal7")
        ("minSpacing" "Metal8")
        ("minSpacing" "Metal9")
)
```

Returns the names of the table spacing constraints defined in the `foundry` constraint group and the layers to which each applies in the technology database identified by `tfID`. The `spacingTables` constraints in this technology database are all single-layer spacing constraints.

techGetViaStackLimit

```
techGetViaStackLimit(
     d_techID
     )
     => l_viaStackLimit / nil
```

Description

Returns the first `viaStackingLimits` constraint in the `foundry` constraint group in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_viaStackLimits	A list showing the via stack limit definition in the foundry constraint group of the specified technology database. If there are multiple via stack limit definitions, the function returns the first definition. The list has the following syntax: `( g_number [ tx_bottomLayer tx_topLayer ] )`
`nil`	The technology database does not exist or no via stack limits are defined.

Example

```
techGetViaStackLimit(tfID)
=> (3  "Metal1"  "Metal5")
```

Returns the first via stack limit defined in the `foundry` constraint group in the technology database identified by `tfID`.

techGetViaStackLimits

```
techGetViaStackLimits(
     d_techID
     )
     => l_viaStackLimits / nil
```

Description

Returns a list of the stacked via limit data defined in the technology database. ASCII technology file location: `foundry` constraint group.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_viaStackLimits	A list of the via stack limits defined in the specified technology database. The list has the following syntax: `( ( g_number [ tx_bottomLayer tx_topLayer ] ) … )`
`nil`	The technology database does not exist or no via stack limits are defined.

Example

```
tfID = techGetTechFile(ddGetObj("newlib"))
db:0x0180200d
```

Returns the database identifier for the current technology database and assigns it to the variable `tfID`.

```
techGetViaStackLimits(tfID)
=> (( 2  "Metal1"  "Metal4"  )
        ( 3   "Metal6"    "Metal8"  )
        ( 2   "Metal5"    "Metal7"  )
   )
```

Returns the via stack limits defined in the `foundry` constraint group of the technology database identified by `tfID`.

techSetViaStackLimits

```
techSetViaStackLimits(
     d_techID
     l_viaStackLimits
     )
     => t / nil
```

Description

Updates the `viaStackingLimits` in the `foundry` constraint group in the specified technology database with one or more sets of `viaStackingLimits` data. This function updates the number for any layer range already in `viaStackingLimits` and adds a new entry for any layer range not already in `viaStackingLimits`. If a `viaStackingLimits` section does not already exist, this function creates one.

Arguments

d_techID	The database identifier of the technology database.
l_viaStackLimits	A list of the via stack limits. The list has the following syntax: `list( list( g_number [ tx_bottomLayer tx_topLayer ] ) … )` where, ■ *g_number* is the maximum number of stacked vias allowed within the layer range. ■ *tx_bottomLayer* is the bottom layer of the layer range. ■ Valid values: The layer name, the layer number ■ *tx_topLayer* is the top layer of the layer range. ■ Valid values: The layer name, the layer number

Value Returned

`t`	The `viaStackingLimits` constraints in the `foundry` constraint group in the specified technology database were updated or created.
`nil`	The technology database does not exist.

Example

```
viaStackingLimits(
  ( 3   "Metal1"    "Metal4"  )
  ( 3   "Metal6"    "Metal8"  )
) ;viaStackingLimits
```

Starts with the above `viaStackingLimits` section in the `foundry` constraint group specifying:

```
techSetViaStackLimits(tfID list(list(2 "Metal1" "Metal4") list(2 "Metal5" "Metal7")))
```

and changes the number for the layer range `Metal1` through `Metal4` from `3` to `2` and adds an entry setting the number for the layer range `Metal5` through `Metal7` to `2`. The technology database then contains the following `viaStackingLimits` section in the `foundry` constraint group:

```
viaStackingLimits(
  ( 2   "Metal1"    "Metal4"  )
  ( 3   "Metal6"    "Metal8"  )
  ( 2   "Metal5"    "Metal7"  )
) ;viaStackingLimits
```

techSetOrderedSpacingRule

```
techSetOrderedSpacingRule(
     d_techID
     t_constraint
     g_value
     tx_layer1
     tx_layer2
     )
     => t / nil
```

Description

Updates the value of the constraint in the specified technology database that (a) is in the `foundry` constraint group, (b) is the first hard constraint with the specified name in an `orderedSpacings` subsection, and (c) is applied to the specified layers. If the specified constraint does not already exist, this function creates a new `orderedSpacings` section containing the constraint.

Arguments

d_techID	The database identifier of the technology database.
t_constraint	The name of the spacing constraint.
	Valid values: Any string (example: `minEnclosure`)
g_value	The value of the spacing constraint.
	Valid values: Any floating-point number, any integer
tx_layer1	The first layer.
	Valid values: The layer name, the layer number
tx_layer2	The second layer.
	Valid values: The layer name, the layer number

Value Returned

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

Example

```
techSetOrderedSpacingRule(tfID "minEnclosure" 0.6 "Metal1" "Via1")
=> t
```

Sets the minimum enclosure constraint for the `Metal1` and `Via1` layers to `0.6` user units in the `foundry` constraint group of the technology database identified by `tfID`.

techSetSpacingRule

```
techSetSpacingRule(
     d_techID
     t_constraint
     g_value
     tx_layer1
     [ tx_layer2 ]
     )
     => t / nil
```

Description

Updates the value of the constraint in the specified technology database that (a) is in the `foundry` constraint group, (b) is the first hard constraint with the specified name in a `spacings` subsection, and (c) is applied to the specified layer or layers. If the specified constraint does not already exist, this function creates a new `spacings` section containing the constraint.

Arguments

d_techID	The database identifier of the technology database.
t_constraint	The name of the spacing constraint.
	Valid values: Any valid spacing constraint name
g_value	The value of the spacing constraint.
	Valid values: Any floating-point number, any integer
tx_layer1	The first layer.
	Valid values: The layer name or number
tx_layer2	The optional second layer.
	Valid values: The layer name or number

Value Returned

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

Example

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

Sets the minimum spacing constraint for `metal1` objects to `0.6` user units in the `foundry` constraint group of the technology database identified by `tfID`.

techSetSpacingRuleTableEntry

```
techSetSpacingRuleTableEntry(
     d_techID
     t_constraintName
     g_index | l_index
     g_value
     tx_layer1
     [ tx_layer2 ]
     )
     => t / nil
```

Description

Updates the specified spacing table in the specified technology database. ASCII technology file location: `spacingTables` subsection in the `foundry` constraint group. If the specified index or index pair is in the table, this function updates the value assigned. If the specified index or index pair is not in the table, this function adds an entry to the table with the specified data.

Arguments

d_techID	The database identifier of the technology database.
t_constraintName	The name of the spacing constraint.
g_index	The index in a one-dimensional table for which to set the table value.
l_index	A list specifying the pair of indexes in a two-dimensional table for which to set the table value. The list has the following syntax: `list( g_index1 g_index2 )` where, ■ *g_index1* is the first table index. ■ *g_index2* is the second table index.
g_value	The value to apply to the specified table index or index pair. Valid values: Any number or string
tx_layer1	The first layer on which the table constraint is applied. Valid values: The layer name, the layer number

tx_layer2	The optional second layer on which the table constraint is applied. Valid values: The layer name, the layer number

Value Returned

`t`	The table was successfully updated.
`nil`	The technology database does not exist or the spacing table does not exist.

Examples

```
techSetSpacingRuleTableEntry(tfID "minSpacing" list(0.005 0.005) 0.14 "Metal1")
=> t
```

Updates the `minSpacing` value to `0.14` in the 2-D `minSpacing` spacing table for `Metal1` containing the indexes `(0.005 0.005)`.

Example 1

The original table:

```
spacingTables(
( minSpacing                "Metal1"
    (( "width"   nil  nil  "length"   nil   nil  ) )
    (
       (0.0005    0.0005    ) 0.12
       (0.0005    0.5605    ) 0.18
       (0.0005    1.5005    ) 0.5
       (0.0005    3.0005    ) 0.9
       (0.0005    7.5005    ) 2.5
       (0.1805    0.0005    ) 0.18
     )
)
```

Changes to:

```
spacingTables(
( minSpacing                "Metal1"
    (( "width"   nil  nil  "length"   nil   nil  ))
    (
       (0.0005    0.0005    ) 0.14
       (0.0005    0.5605    ) 0.18
       (0.0005    1.5005    ) 0.5
       (0.0005    3.0005    ) 0.9
       (0.0005    7.5005    ) 2.5
       (0.1805    0.0005    ) 0.18
     )
  )
```

Example 2

```
techSetSpacingRuleTableEntry(tfID "minSpacing" list(0.1 0.005) 0.14 "Metal1")
=> t
```

The original table:

```
spacingTables(
( minSpacing                  "Metal1"
    (( "width"   nil  nil  "length"   nil   nil  ) )
    (
       (0.0005    0.0005    ) 0.12
       (0.0005    0.5605    ) 0.18
       (0.0005    1.5005    ) 0.5
       (0.0005    3.0005    ) 0.9
       (0.0005    7.5005    ) 2.5
       (0.1805    0.0005    ) 0.18
      )
)
```

changes to:

```
spacingTables(
( minSpacing                  "Metal1"
    (( "width"   nil  nil  "length"   nil   nil  ))
    (
       (0.0005    0.0005    ) 0.12
       (0.0005    0.5605    ) 0.18
       (0.0005    1.5005    ) 0.5
       (0.0005    3.0005    ) 0.9
       (0.0005    7.5005    ) 2.5
        (0.1000    0.005     ) 0.14
       (0.1805    0.0005    ) 0.18
      )
  )
```

techSetViaStackLimit

```
techSetViaStackLimit(
     d_techID
     g_number
     [ tx_bottomLayer tx_topLayer ]
     )
     => t / nil
```

Description

Updates a `viaStackingLimits` constraint in the `foundry` constraint group in the specified technology database with one set of via stack limit data. If the specified layer range is not already specified in the database, this function creates a new constraint with the data. If the `viaStackingLimits` subsection does not exist in the `foundry` constraint group, this function creates it with the specified data.

Arguments

d_techID	The database identifier of the technology database.
g_number	The maximum number of stacked vias allowed within the layer range. Valid values: Any positive integer
tx_bottomLayer	The bottom layer of the layer range. Valid values: Any routing layer name
tx_topLayer	The top layer of the layer range. Valid values: Any routing layer name

Value Returned

`t`	The `viaStackingLimits` subsection in the specified technology database was updated or created.
`nil`	The technology database does not exist.

Example

```
techSetViaStackLimit(tfID 4  "Metal1"  "Metal5")
=> t
```

Creates the following in a technology database with no `viaStackingLimits` subsection in the `foundry` constraint group:

```
viaStackingLimits(
 ( 4   "Metal1"    "Metal5"  )
) ;viaStackingLimits
```

Then,

```
techSetViaStackLimit(tfID 3  "Metal1"  "Metal5")
=> t
```

changes the stack limit:

```
viaStackingLimits(
 ( 3   "Metal1"    "Metal5"  )
) ;viaStackingLimits
```

Then,

```
techSetViaStackLimit(tfID 2  "Metal6"  "Metal8")
=> t
```

Adds another constraint:

```
viaStackingLimits(
 ( 3   "Metal1"    "Metal5"  )
 ( 3   "Metal6"    "Metal8"  )
) ;viaStackingLimits
```

And,

```
techSetViaStackLimit(tfID 5)
```

adds yet another constraint:

```
viaStackingLimits(
 ( 3   "Metal1"    "Metal5"  )
 ( 3   "Metal6"    "Metal8"  )
 ( 5  )
) ;viaStackingLimits
```

Electrical Constraints and Layer Attributes Functions

The electrical constraints and layer attributes functions set and get electrical constraints.

techCreateElectricalRuleTable

```
techCreateElectricalRuleTable(
     d_techID
     t_constraintName
     l_indexDefinitions
     tx_layer1
     [ tx_layer2 ]
     )
     => t / nil
```

Description

Creates a current density table and indexes for the named layer attribute in the specified technology database. ASCII technology file location: `currentDensityTables` subsection of the `layerRules` section. To create table entries, use `techSetElectricalRuleTableEntry`. If a `currentDensityTables` subsection does not exist, this function creates one. If a current density table already exists for the named attribute applied to the specified layer or layers, this function replaces it; the original table entries are deleted.

Arguments

d_techID	The database identifier of the technology database.
t_constraintName	The name of the electrical constraint. Valid values: `peakACCurrentDensity`, `avgACCurrentDensity`, `rmsACCurrentDensity`, `avgDCCurrentDensity`
l_indexDefinitions	A list defining the name of the index or indexes for the table. The list has the following syntax: `list( [ t_index1Name ] nil nil [ t_index2Name ] nil nil )` where, ■ *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 ■ *t_index2Name* is the name of the second dimension of a two-dimensional table. Valid values: Any number or string
tx_layer1	The first layer on which to apply the table constraint. Valid values: The layer name, the layer number
tx_layer2	The optional second layer on which to apply the table constraint. Valid values: The layer name, the layer number

Value Returned

`t`	The current density table was created.
`nil`	The current density table was not created.

Example

```
techCreateElectricalRuleTable(tfID "peakACCurrentDensity"
```

```
list("frequency" nil nil  "width" nil nil) "Metal1")
=> t
```

Creates the table electrical rule `peakACCurrentDensity` with the indexes `frequency` and `width` applied to layer `Metal1` in the technology database identified by `tfID`.

techGetCurrentDensityRules

```
techGetCurrentDensityRules(
     d_techID
     )
     => l_currentDensityRules / nil
```

Description

Returns a list of the current density attributes for layers from the specified technology database.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_currentDensityRules	A list of the current density constraints for layers in the technology database.
nil	The technology database does not exist or no current density constraints are specified for any layers in the technology database.

Example

```
techGetCurrentDensityRules (tfID)
=> (("peakACCurrentDensity" 2.3 "Metal1")
    ("avgACCurrentDensity" 5.0 "Metal1")
    ("rmsACCurrentDensity" 4.1 "Metal1")
    ("avgDCCurrentDensity" 3.0 "Metal1")
   )
```

Returns a list of the current density rules specified for the layers in the technology database identified by `tfID`.

techGetCurrentDensityRuleTable

```
techGetCurrentDensityRuleTable(
     d_techID
     t_ruleTableName
     tx_layer
     )
     => l_currentDensityRuleTable / nil
```

Description

Returns the named current density attribute table for the specified layer in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
t_ruleTableName	The name of the constraint table to return.
tx_layer	The layer for which to return the constraint table. Valid values: The layer name, layer number

Value Returned

l_currentDensityRuleTable	The current density constraint table.
nil	The technology database does not exist or the requested current density constraint table does not exist.

Example

```
techGetCurrentDensityRuleTable (tfID "avgACCurrentDensity" "metal1")
=> (("frequency" nil nil "width" nil nil)
    (10.0 0.25) 2.3e-07
    (10.0 0.35) 3.5e-07
    (210.0 0.25) 3.5e-07
    (210.0 0.35) 3.5e-07
   )
```

Returns the requested current density constraint table in the technology database identified by `tfID`.

techGetCurrentDensityRuleTables

```
techGetCurrentDensityRuleTables(
     d_techID
     )
     => l_currentDensityRuleTables / nil
```

Description

Returns a list of the current density attribute tables for layers in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_currentDensityRuleTables	A list of the current density constraint tables for layers in the technology database.
nil	The technology database does not exist or no current density constraint tables are specified for any layers in the technology database.

Example

```
techGetCurrentDensityRuleTables (tfID)
=> (("peakACCurrentDensity" "Metal1")
    ("avgACCurrentDensity" "Metal1")
    ("rmsACCurrentDensity" "Metal1")
   )
```

Returns a list of the current density constraint tables specified for the layers in the technology database identified by `tfID`.

techGetElectricalRules

```
techGetElectricalRules(
     d_techID
     )
     => l_electricalRules / nil
```

Description

Returns a list of all of the current density layer attributes and antenna models constraints defined in the specified technology database. ASCII technology file location: `currentDensity` subsection of the `layerRules` section and the `antennaModels` section of the `foundry` constraint group.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_electricalRules	A list of lists containing the electrical attribute and antenna models settings. The following syntax is used to return current density layer attributes: `( ( t_constraint g_value lt_layer1 [ lt_layer2 ] ) … )` where, ■ *t_constraint* is the name of the electrical constraint. ■ *g_value* is the value of the electrical constraint. ■ *lt_layer1* is the first layer. ■ *lt_layer2* is the second layer. Only returned for two-layer constraints.

The following syntax is used to return antenna model constraints:

```
("nameAntennaRule"
 ( l_areaRatio
 l_sideAreaRatio
 l_l_diffAreaRatio
 l_diffSideAreaRatio
 l_cumAreaRatio
 l_cumDiffAreaRatio
 l_cumSideAreaRatio
 l_cumDiffSideAreaRatio
 nil
 nil)
t_layerName )
```

where,

- *name* is the antenna oxide model name: `default`, `second`, `third`, or `fourth`
- *l_areaRatio* is the no-side area ratio.
- *l_sideAreaRatio* is the side area ratio.
- *l_diffAreaRatio* is the no-side diffusion area ratio.
- *l_diffSideAreaRatio* is the side diffusion area ratio.
- *l_cumAreaRatio* is the no-side cumulative area ratio.
- *l_cumDiffAreaRatio* is the no-side cumulative diffusion area ratio.
- *l_cumSideAreaRatio* is the side cumulative area ratio.
- *l_cumDiffSideAreaRatio* is the side cumulative diffusion area ratio.

`nil` — The technology database does not exist or there are no electrical layer attributes or antenna models defined.

Example

```
techGetElectricalRules(tfID)
=> (  ("defaultAntennaRule"
       (475.0 nil nil nil 1200.0
           ((0.0 1200.0)
           (0.099 1200.0)
```

```
          (0.1 55750.0)
          (1.0 62500.0)
          ) nil nil nil nil
       ) "Metal1"
   )
…
   ("peakACCurrentDensity" 2.3 "Metal1")
   ("avgACCurrentDensity" 5.0 "Metal1")
   ("rmsACCurrentDensity" 4.1 "Metal1")
   ("avgDCCurrentDensity" 3.0 "Metal1")
)
```

Returns the electrical layer attributes and constraints defined in the technology database identified by `tfID`.

The antenna rule shown contains the following data:

- antenna oxide model name: `default`
- no-side area ratio: `475.0`
- side area ratio: `nil`
- no-side diffusion area ratio: `nil`
- side diffusion area ratio: `nil`
- no-side cumulative area ratio: `12000.0`
- no-side cumulative diffusion area ratio:

```
((0.0 1200.0)
          (0.099 1200.0)
          (0.1 55750.0)
          (1.0 62500.0)
          )
```

- side cumulative area ratio: `nil`
- side cumulative diffusion area ratio: `nil`

techGetElectricalRuleTable

```
techGetElectricalRuleTable(
     d_techID
     t_constraint
     tx_layer1
     [ tx_layer2 ]
     )
     => l_tables / nil
```

Description

Returns the table data for the specified layer attribute defined in the specified technology database. ASCII technology file location: `currentDensityTables` subsection of the `layerRules` section. Preferred: Use `techGetCurrentDensityRuleTable` function.

Arguments

d_techID	The database identifier of the technology database.
t_constraint	The name of the current density constraint.
tx_layer1	The first layer on which to apply the table constraint. Valid values: The layer name, the layer number
tx_layer2	The optional second layer on which to apply the table constraint. Valid values: The layer name, the layer number

Value Returned

l_tables — A list defining the table entries. The list has the following syntax:

```
( ( l_indexDefinitions ) l_table )
```

where,

- *l_indexDefinitions* is a list naming the table indexes. This list has the following syntax:

  ```
  ( nt_index1Name  nil nil
  [ nt_index2Name  nil nil] )
  ```

  where, *nt_index1Name* is the name of the first dimension of a two-dimensional table or the only dimension of a one-dimensional table and *nt_index2Name* is the name the second dimension of a two-dimensional table.

- *l_table* is a list of the table entries.

`nil` — The technology database does not exist or there are no current density tables defined.

Example

```
techGetElectricalRuleTable(tfID "peakACCurrentDensity" "Metal2")
=> ((("frequency" nil nil "width" nil
                nil
            ) 5.5e-06
        )
        (1.0 0.3) 5e-06
        (1.0 0.4) 5.5e-06
        (2e+07 0.3) 5.5e-06
        (2e+07 0.4) 6e-07
   )
```

Returns the table data for the `peakACCurrentDensity` constraint applied to layer `Metal2` in the `layerRules` section group of the technology database identified by `tfID`.

techGetElectricalRuleTableEntry

```
techGetElectricalRuleTableEntry(
     d_techID
     t_constraint
     l_index
     tx_layer1
     [ tx_layer2 ]
     )
     => g_value / nil
```

Description

Returns the value of the specified index in the specified current density table in the specified technology database. ASCII technology file location: `currentDensityTables` subsection of the `layerRules` section.

Arguments

d_techID	The database identifier of the technology database.
t_constraint	The name of the constraint.
l_index	The list of indexes. The list has the following syntax: `( g_index1 [ g_index2 ] )` where, ■ *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.
tx_layer1	The first layer on which the table constraint is applied. Valid values: The layer name, the layer number
tx_layer2	The optional second layer on which the table constraint is applied. Valid values: The layer name, the layer number

Value Returned

g_value	The value of the specified index.
nil	The technology database does not exist or the table constraint is not defined.

Example

```
techGetElectricalRuleTableEntry(tfID "peakACCurrentDensity" list(1.0 0.3)
"Metal2")
=> 5e-08
```

Returns the requested entry from the current density constraint table in the technology database identified by `tfID`.

techGetElectricalRuleTables

```
techGetElectricalRuleTables(
     d_techID
     )
     => l_tables / nil
```

Description

Returns a list of the current density tables defined in the specified technology database. ASCII technology file location: `currentDensityTables` subsection in the `layerRules` section. Preferred: Use techGetCurrentDensityRuleTables function.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_tables	A list of the current density table constraint names and the layers to which they apply. The list has the following syntax: `( ( t_constraint tx_layer1 [ tx_layer2 ] ) … )` where, ■ *t_constraint* is the name of the electrical constraint. Valid values: Any string ■ *tx_layer1* is the first layer on which the table constraint is applied. Valid values: The layer name, the layer number ■ *tx_layer2* is the optional second layer on which the table constraint is applied. Valid values: The layer name, the layer number
`nil`	The technology database does not exist or there are no current density tables defined.

Example

```
techGetElectricalRuleTables(tfID)
=> (("peakACCurrentDensity" "Metal1")
       ("peakACCurrentDensity" "Via1")
       ("peakACCurrentDensity" "Metal2")
       ("peakACCurrentDensity" "Via2")
   )
```

techGetOrderedElectricalRules

```
techGetOrderedElectricalRules(
     d_techID
     )
     => l_electricalRules / nil
```

Description

Returns a list of all of the ordered electrical constraints specified in the technology database. ASCII technology file location: `orderedElectrical` constraints in the `foundry` constraint group. This is a legacy section of the technology file used to store data converted from a technology file containing an obsolete subclass.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_electricalRules	A list of lists containing the ordered electrical constraint settings. The list has the following syntax: `( ( t_constraint g_value lt_layer1 lt_layer2 ) … )` where, ■ *t_constraint* is the name of the ordered electrical constraint. ■ *g_value* is the value of the ordered electrical constraint. ■ *lt_layer1* is the first layer. ■ *lt_layer2* is the second layer.
`nil`	The technology database does not exist or there are no ordered electrical constraints defined.

Example

```
techGetOrderedElectricalRules(tfID)
=> ( parallelCap 2.00 metal1  metal2 )
```

```
( parallelCap 2.00 metal3  metal4 )
```

Returns the ordered electrical constraints defined in the technology database identified by `tfID`.

techSetElectricalRule

```
techSetElectricalRule(
     d_techID
     t_constraint
     g_value
     tx_layer1
     [ tx_layer2 ]
     )
     => t / nil
```

Description

Updates the value of the layer attribute in the specified technology database that (a) is the first layer attribute with the specified name in a `currentDensity` subsection of the `layerRules` section, and (b) is applied to the specified layer or layers. If the specified attribute does not already exist, this function creates a new `currentDensity` subsection containing the layer attribute. You cannot update or add `antennaModels` constraints with this function.

Arguments

d_techID	The database identifier of the technology database.
t_constraint	The name of the layer attribute.
	Valid values: `peakACCurrentDensity`, `avgACCurrentDensity`, `rmsACCurrentDensity`, `avgDCCurrentDensity`
g_value	The value of the layer attribute.
	Valid values: Any floating-point number, any integer
tx_layer1	The first layer.
	Valid values: The layer name, the layer number
tx_layer2	The optional second layer.
	Valid values: The layer name, the layer number

Value Returned

`t`	The `currentDensity` layer attribute was successfully updated or created.
`nil`	The technology database does not exist or the layers are not defined.

Example

```
techSetElectricalRule(tfID "peakACCurrentDensity" 0.6 "metal1")
=> t
```

Sets the peak AC current density for `Metal1` to `0.6` user units in the technology database identified by `tfID`.

techSetElectricalRuleTableEntry

```
techSetElectricalRuleTableEntry(
     d_techID
     t_constraint
     l_index
     g_value
     tx_layer1
     [ tx_layer2 ]
     )
     => t / nil
```

Description

Updates the specified current density table in the specified technology database. ASCII technology file location: `currentDensityTables` subsection of the `layerRules` section. If the specified index or index pair is in the table, this function updates the value assigned. If the specified index or index pair is not in the table, this function adds an entry to the table with the specified data.

Arguments

d_techID	The database identifier of the technology database.
t_constraint	The name of the current density constraint.
l_index	The list of indexes. The list has the following syntax: `( g_index1 [ g_index2 ] )` where, ■ *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 apply to the specified table index or index pair. Valid values: Any number or string
tx_layer1	The first layer on which the table constraint is applied. Valid values: The layer name, the layer number
tx_layer2	The optional second layer on which the table constraint is applied. Valid values: The layer name, the layer number

Value Returned

`t`	The table was successfully updated.
`nil`	The technology database does not exist or the current density table does not exist.

Example

```
techSetElectricalRuleTableEntry(tfID "peakACCurrentDensity" list(1.0 0.3) 5e-08
"Metal2))
=> t
```

Changes the original table below:

```
currentDensityTables(
( "peakACCurrentDensity"   "Metal2"
    (("frequency" nil nil "width" nil nil))
       (
         (1.0 0.3)      5e-05
         (1 0. 0.4)     5e-06
         (2e+07 0.3)    5.5e-06
         (2e+07 0.4)     6e-07
       )
)
```

to:

```
currentDensityTables(
( "peakACCurrentDensity"   "Metal2"
    (("frequency" nil nil "width" nil nil))
       (
         (1.0 0.3)      5e-08
         (1 0. 0.4)     5e-06
         (2e+07 0.3)    5.5e-06
         (2e+07 0.4)     6e-07
       )
)
```

techSetOrderedElectricalRule

```
techSetOrderedElectricalRule(
     d_techID
     t_constraint
     g_value
     tx_layer1
     tx_layer2
     )
     => t / nil
```

Description

Updates the value of the first ordered electrical constraint for the specified layers in the specified technology database. ASCII technology file location: `orderedElectrical` subsection in the `foundry` constraint group. This is a legacy section of the technology file used to store data converted from a technology file containing the obsolete `orderedCharacterizationRules` subclass.

Arguments

d_techID	The database identifier of the technology database.
t_constraint	The name of the ordered electrical constraint.
	Valid values: Any string
g_value	The value of the electrical constraint.
	Valid values: Any floating-point number, any integer
tx_layer1	The first layer.
	Valid values: The layer name, the layer number
tx_layer2	The second layer.
	Valid values: The layer name, the layer number

Value Returned

`t`	The ordered electrical constraint was updated or created successfully.
`nil`	The technology database does not exist or the layers are not defined.

Example

```
techSetOrderedElectricalRule(tfID "minEnclosure" 0.6 "via" "metal1")
=> t
```

Sets the minimum enclosure constraint for the `via` and `metal1` layers to `0.6` user units in the technology database identified by `tfID`.

5

Place and Route Functions

The place and route functions are used to manipulate place and route technology data.

The tech*Pr* functions are used for backward compatibility. Before using these functions you need to ensure that the `LEFDefaultRouteSpec` ***constraint group is defined in your technology database. Otherwise, you may get incorrect results. Therefore, it is not recommended to use these functions.***

However, a new SKILL API is introduced starting IC6.1.4 release (functions starting with cst*) which should be used.

techSetPrRoutingLayers

```
techSetPrRoutingLayers(
     d_techID
     l_routingLayers
     )
     => t / nil
```

Description

Updates the specified technology database to change or add the preferred routing direction for each of the specified layers. ASCII technology file location: `routingDirections` subsection in the `layerRules` section.

Arguments

d_techID	The database identifier of the technology database.
l_routingLayers	A list of the layers and the direction keywords you want to define. The list has the following syntax: `list ( list ( tx_layer t_direction ) … )` where, ■ *tx_layer* is the layer. Valid values: The layer name, the layer number ■ *t_direction* is the preferred routing direction for the layer. Valid values: `horizontal`, `vertical`

Value Returned

`t`	The specified technology database was updated.
`nil`	The technology database does not exist.

Example

```
techSetPrRoutingLayers(tfID list(list("METAL2" "vertical")
    list("METAL3" "vertical")))
=> t
```

Changes or adds the preferred routing direction for `METAL2` and `METAL3` to `vertical` in the `routingDirections` subsection of the technology database identified by `tfID`.

techSetPrRoutingLayer

```
techSetPrRoutingLayer(
     d_techID
     tx_routingLayer
     t_direction
     )
     => t / nil
```

Description

Updates the specified technology database to change or add the preferred routing direction for the specified layer. ASCII technology file location: `routingDirections` subsection in the `layerRules` section.

Arguments

d_techID	The database identifier of the technology database.
tx_routinglayer	The layer for which to update the direction.
	Valid values: The layer name, the layer number
t_direction	The preferred routing direction for the layer.
	Valid values: `horizontal`, `vertical`

Value Returned

`t`	The data was added to the specified technology database.
`nil`	The technology database does not exist.

Example

```
techSetPrRoutingLayer(tfID "metal9" "horizontal")
=> t
```

Changes or adds the preferred routing direction for `metal9` to `horizontal` in the `routingDirections` subsection of the technology database identified by `tfID`.

techGetPrRoutingLayers

```
techGetPrRoutingLayers(
     d_techID
     )
     => l_routingLayers / nil
```

Description

Returns a list of layers and their preferred routing directions. In the ASCII technology file, the layers are defined in the `validLayers` constraint in the `interconnect` subsection of the `LEFDefaultRouteSpec` constraint group. The preferred routing directions of the layers are specified in the `routingDirections` subsection of the `layerRules` section.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_routingLayers	A list of the layers and their routing directions defined in the specified technology database. The list has the following syntax: `( ( t_layer t_direction ) … )` where, ■ *t_layer* is the layer name. ■ *t_direction* is the direction assigned to the layer.
nil	The technology database does not exist, or the `validLayers` constraint is not defined in the `LEFDefaultRouteSpec` constraint group.

Example

```
techGetPrRoutingLayers(tfID)
=> (("METAL1" "horizontal")
    ("METAL2" "vertical")
    ("METAL3" "horizontal")
    ("METAL4" "vertical")
    ("METAL5" "horizontal")
    ("METAL6" "vertical")
    ("METAL7" "horizontal")
```

```
    ("METAL8" "vertical")
  )
```

Returns the routing layers and their directions as defined in the technology database identified by `tfID`.

techGetPrRoutingDirection

```
techGetPrRoutingDirection(
     d_techID
     tx_layer
     )
     => t_direction / nil
```

Description

Returns the direction assigned to the specified routing layer technology database. ASCII technology file location: `routingDirections` subsection in the `layerRules` section.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The layer for which you want to get the usage. Valid values: The layer name, the layer number

Value Returned

t_direction	The preferred direction assigned to the specified layer. If the layer is listed but no direction is assigned, returns `none`.
`nil`	The technology database does not exist, the layer is not a routing layer, or no direction is specified for the layer.

Example

```
techGetPrRoutingDirection(tfID "METAL6")
=> "vertical"
```

Returns the direction (`vertical`) assigned to the layer `METAL6` in the technology database identified by `tfID`.

techIsPrRoutingLayer

```
techIsPrRoutingLayer(
     d_techID
     tx_layer
     )
     => t / nil
```

Description

Indicates whether the specified layer is a routing layer, assigned a routing direction in the specified technology database. ASCII technology file location: `routingDirections` subsection of the `layerRules` section.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The layer to check. Valid values: The layer name, the layer number

Value Returned

`t`	The specified layer is a routing layer in the specified technology database.
`nil`	The technology database does not exist or the layer is not defined in the technology database or it is not a routing layer.

Example

```
techIsPrRoutingLayer(techID "metal2")
=> t
```

The `metal2` layer is a routing layer in the technology database identified by `techID`.

techSetPrViaTypes

```
techSetPrViaTypes(
     d_techID
     l_viaTypes
     )
     => t / nil
```

Description

Adds the specified via definitions for the specified cell and cellview to the list of valid vias for the specified constraint group in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
l_viaTypes	A list of the via definitions and the optional constraint group names you want to define. The list has the following syntax: `list ( list ( list ( t_viaDefName t_view ) [ t_constraintGroupName ] ) … )` where, ■ *t_viaDefName* is the cell name of the device (the *viaDefName* specified in the `viaDefs` section). ■ *t_view* is the view name of the device. ■ *t_constraintGroupName* is the name of the constraint group.

Value Returned

`t`	The via definitions were added.
`nil`	The technology database does not exist or one or more cell name/cellview combinations are not defined.

Example

```
techSetPrViaTypes(techID list(
(list("via1" "via") "default")
(list("via2" "via") "default")
)
```

```
)
=> t
```

techSetPrViaType

```
techSetPrViaType(
     d_techID
     l_viaDef
     t_constraintGroupName
     )
     => t / nil
```

Description

Adds the specified via definition to the specified technology database. ASCII technology file location: `validVias` section of the specified constraint group.

Arguments

d_techID	The database identifier of the technology database.
l_viaDef	A list containing the via definition name and its associated view name. The list has the following syntax: `list ( t_viaDefName t_view )` where, ■ *t_viaDefName* is the name of the via definition to add to `validVias` in the specified constraint group. ■ *t_view* is the name of the view associated with the via definition. Valid values: For a standard via definition, always `layout`. For a custom via definition, the view specified in the via definition.
t_constraintGroupName	The name of the constraint group.

Value Returned

`t`	The via definitions were added.
`nil`	The technology database does not exist or there is no via definition with the specified name and view.

Example

```
techSetPrViaType(tfID list("M2_M1" "via") "default")
=> t
```

Adds the custom via definition `M2_M1` with view `via` to the `default` constraint group in the technology database identified by `tfID`.

To add the standard via definition `M1_Poly1` to the constraint group named `constGp1`,

```
techSetPrViaType(tfID list("M2_Poly1" "layout") "constGp1")
=> t
```

techGetPrViaTypes

```
techGetPrViaTypes(
     d_techID
     )
     => l_viaTypes / nil
```

Description

Returns a list of the vias defined in the specified technology database. ASCII technology file location: `validVias` sections.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_viaTypes	A list containing the name and view of each via definition and the constraint group each is in. The list has the following syntax: `( ( ( t_viaDefName t_view ) t_constraintGroupName )… )` where, ■ *t_viaDefName* is the name of the via definition. ■ *t_view* is the name of the view. (Standard via definitions always have the view `layout`.) ■ *t_constraintGroupName* is the name of the constraint group.
`nil`	The technology database does not exist or it contains no `validVias` subsection in any constraint group.

Example

```
techGetPrViaTypes(tfID)
((("viad" "via") "default")
(("viap" "via") "default")
(("via1a" "via") "default")
(("via2" "via") "default")
(("via3" "via") "default")
(("via4" "via") "default")
```

```
(("via5" "via") "default")
(("via6" "via") "default")
(("via7" "via") "default")
(("via1_west" "via") "default")
(("via1_east" "via") "default")
(("via2_south" "via") "default")
(("via2_north" "via") "default")
(("via3_west" "via") "default")
(("via3_east" "via") "default")
(("via4_south" "via") "default")
(("via4_north" "via") "default")
(("via5_west" "via") "default")
(("via5_east" "via") "default")
(("via7_west" "via") "default")
(("via7_east" "via") "default")
(("via1a_1x2" "via") "default")
(("via1a_2x1" "via") "default")
(("via2_1x2" "via") "default")
(("via2_2x1" "via") "default")
(("via3_1x2" "via") "default")
(("via3_2x1" "via") "default")
(("via4_1x2" "via") "default")
(("via4_2x1" "via") "default")
(("via5_1x2" "via") "default")
(("via5_2x1" "via") "default")
(("via6_1x2" "via") "default")
(("via6_2x1" "via") "default")
(("via7_1x2" "via") "default")
(("via7_2x1" "via") "default")
(("myVia5" "via") "myRoute")
(("myVia1a" "via") "myRoute")
(("myVia2a" "via") "myRoute"))
```

techGetPrViaType

```
techGetPrViaType(
     d_techID
     l_viaDef
     )
     => t_viaType / nil
```

Description

Returns all constraint groups in which the specified via definition is listed as a valid via in the specified technology database. ASCII technology file location: `validVias` section.

Arguments

d_techID	The database identifier of the technology database.
l_viaDef	A list containing the via definition name and its associated view name. The list has the following syntax: `list ( t_viaDefName t_view )` where, ■ *t_viaDefName* is the name of the via definition to add to `validVias` in the specified constraint group. ■ *t_view* is the name of the view associated with the via definition. (Standard via definitions always have the view `layout`.)

Value Returned

t_viaType	The name of the constraint group.
`nil`	The technology database does not exist or the via definition does not appear in the `validVias` list in any constraint group.

Example

```
techGetPrViaType(tfID list("viad" "via"))
=> ("CG__0" "default" "myRoute" "myRoute1")
```

Returns the constraint groups with the requested via definition in their `validVias` section in the technology database identified by `tfID`.

techIsPrViaDevice

```
techIsPrViaDevice(
     d_techID
     l_viaDef
     )
     => t / nil
```

Description

Indicates whether the specified via definition is listed in the specified technology database. ASCII technology file location: in any `validVias` section the specified technology file.

Arguments

d_techID	The database identifier of the technology database.
l_viaDef	A list containing the via definition name and its associated view name. The list has the following syntax: `list ( t_viaDefName t_view )` where, ■ *t_viaDefName* is the name of the via definition to add to `validVias` in the specified constraint group. ■ *t_view* is the name of the view associated with the via definition. (Standard via definitions always have the view `layout`.)

Value Returned

`t`	The specified via definition is listed in a `validVias` section in at least one constraint group in the specified technology database.
`nil`	The technology database does not exist or the via is not listed in any `validVias` section in the technology database.

Example

```
techIsPrViaDevice(tfID list("viad" "via"))
=> t
```

The custom via definition `viad` with view `via` is listed in at least one `validVias` section in the technology database identified by `tfID`.

techSetPrStackVias

```
techSetPrStackVias(
     d_techID
     l_stackVias
     )
     => t / nil
```

Description

Updates the specified technology database to mark the listed pairs of via layers stackable. ASCII technology file location: `stackable` subsection in the `spacings` section of the `foundry` constraint group for the specified via layer pairs.

Arguments

d_techID	The database identifier of the technology database.
l_stackVias	A list of lists of two via layers. The list has the following syntax: `list ( list ( tx_viaLayer1 tx_viaLayer2 ) … )` where, ■ *tx_viaLayer1* is one of the layers in the via layer pair. Valid values: The layer name or the layer number of a layer with the layer function `cut` or `li` ■ *tx_viaLayer2* is the second layer of the pair. Valid values: The layer name or the layer number of a layer with the layer function `cut` or `li`

Value Returned

`t`	The data was added to the specified technology database.
`nil`	The technology database does not exist or one or more of the specified layers are not defined.

Example

```
techSetPrStackVias(tfID
list(list("via" "via2") list("via3" "via4"))
```

```
)
=> t
```

Specifies `via1` and `via2` stackable and `via3` and `via4` stackable in the technology database identified by `tfID`. Adds the following to the foundry constraint group:

```
spacings(
         (stackable     "via"    "via2"   t)
         (stackable     "via3"   "via4"   t)
        ) ;spacings
```

techSetPrStackVia

```
techSetPrStackVia(
     d_techID
     tx_viaLayer1
     tx_viaLayer2
     )
     => t / nil
```

Description

Updates the specified technology database to mark the specified pair of via layers stackable in the foundry constraint group. If the stackable section does not already exist, this function creates it.

Arguments

d_techID	The database identifier of the technology database.
tx_viaLayer1	One of the layers in the via layer pair. Valid values: The layer name or the layer number of a layer with the layer function `cut` or `li`
tx_viaLayer2	The second layer in the via layer pair. Valid values: The layer name or the layer number of a layer with the layer function `cut` or `li`

Value Returned

`t`	The data was added to the `foundry` constraint group in the specified technology database.
`nil`	The technology database does not exist or one or more of the specified layers are not defined.

Example

```
techSetPrStackVia(tfID "via" "via2")
=> t
```

Defines `via` and `via2` as stackable in the technology database identified by `tfID`. Adds the following to the foundry constraint group:

```
spacings(
```

```
 (stackable     "via"    "via2"   t)
) ;spacings
```

techGetPrStackVias

```
techGetPrStackVias(
     d_techID
     )
     => l_stackVias / nil
```

Description

Returns a list of the via layer pairs defined as stackable in the specified technology database. ASCII technology file location: `stackable` in the `foundry` constraint group of specified technology file.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_stackVias	A list of lists of the via layer pairs that are defined as stackable in the technology database. The list has the following syntax: `list ( list ( lt_layer1 lt_layer2 ) … )` where, ■ *lt_layer1* is the first layer of a pair. ■ *lt_layer2* is the second layer of a pair.
`nil`	The technology database does not exist or no `stackable` via layers are defined in the `foundry` constraint group.

Example

```
techGetPrStackVias(tfID)
=> (("via" "via2")
    ("via3" "via4")
   )
```

Returns the via layer pairs defined as stackable in the `foundry` constraint group of the technology database identified by `tfID`.

techIsPrStackVia

```
techIsPrStackVia(
     d_techID
     tx_viaLayer1
     tx_viaLayer2
     )
     => t / nil
```

Description

Indicates whether the specified via layer pair is defined as stackable in the specified technology database. ASCII technology file location: `foundry` constraint group.

Arguments

d_techID	The database identifier of the technology database.
tx_viaLayer1	The first layer in the via layer pair. Valid values: The layer name, the layer number
tx_viaLayer2	The second layer in the via layer pair. Valid values: The layer name, the layer number

Value Returned

`t`	The specified via layer pair is defined as stackable in the `foundry` constraint group in the specified technology database.
`nil`	The technology database does not exist or the via layer pair is not defined as stackable in the `foundry` constraint group in the technology database.

Example

```
techIsPrStackVia(tfID "via" "via2")
=> t
```

The `via`/`via2` layer pair is defined as stackable in the technology database identified by `tfID`.

techSetPrMastersliceLayers

```
techSetPrMastersliceLayers(
     d_techID
     l_msLayers
     )
     => t / nil
```

Description

Verifies that the specified layers in the specified technology database are valid masterslice layers. A masterslice layer is any layer that meets both of the following conditions:

1. The layer is explicitly assigned one of the following functions: `nwell`, `pwell`, `ndiff`, `pdiff`, or `poly`. ASCII technology file location: in the `functions` subsection of the `layerRules` section.

2. If the technology database contains a `LEFDefaultRouteSpec` constraint group, the layer is not listed as a valid layer in the `validLayers` section of that constraint group.

You can define layers that are not valid layers in a `LEFDefaultRouteSpec` constraint group to be masterslice layers by assigning them the proper functions with `techSetLayerFunction` or `techSetLayerFunctions`. For more information about the `functions` subsection of the technology file, see functions in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
l_msLayers	A list of the layers that you want to verify are masterslice layers. The list has the following syntax: `list ( tx_msLayer … )` where, *tx_msLayer* is a layer to verify as a masterslice layer. Valid values: The layer name, the layer number

Value Returned

`t`	The specified layers in the specified technology database are valid masterslice layers.
`nil`	One or more of the specified layers does not meet the requirements for valid masterslice layers or the technology database does not exist.

Example

```
techSetPrMastersliceLayers(tfID list("diff" "poly"))
=> t
```

Verifies that the layers `diff` and `poly` in the technology database identified by `tfID` meet the requirements for valid masterslice layers.

techSetPrMastersliceLayer

```
techSetPrMastersliceLayer(
     d_techID
     tx_msLayer
     )
     => t / nil
```

Description

Verifies that the specified layer in the specified technology database is a valid masterslice layer. A masterslice layer is any layer that meets both of the following conditions:

1. The layer is explicitly assigned one of the following functions: `nwell`, `pwell`, `ndiff`, `pdiff`, or `poly`. ASCII technology file location: in the `functions` subsection of the `layerRules` section.

2. If the technology database contains a `LEFDefaultRouteSpec` constraint group, the layer is not listed in the `validLayers` section of that constraint group.

You can define a layer that is not a valid layer in a `LEFDefaultRouteSpec` constraint group to be a masterslice layer by assigning it the proper function with `techSetLayerFunction`. For more information about the `functions` subsection of the technology file, see functions in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
tx_msLayer	The layer you want to verify is a masterslice layer. Valid values: The layer name, the layer number

Value Returned

`t`	The specified layer in the specified technology database is a valid masterslice layer.
`nil`	The specified layer does not meet the requirements for a valid masterslice layer or the technology database does not exist.

Example

```
techSetPrMastersliceLayer(tfID "poly")
=> t
```

Verifies that the layer `poly` in the technology database identified by `tfID` meets the requirements for valid masterslice layers.

techGetPrMastersliceLayers

```
techGetPrMastersliceLayers(
     d_techID
     )
     => l_msLayers / nil
```

Description

Returns a list of the layers in the specified technology database that are valid masterslice layers. A masterslice layer is any layer that meets both of the following conditions:

1. The layer is explicitly assigned one of the following functions: `nwell`, `pwell`, `ndiff`, `pdiff`, or `poly`. ASCII technology file location: in the `functions` subsection of the `layerRules` section.

2. If the technology database contains a `LEFDefaultRouteSpec` constraint group, the layer is not listed in the `validLayers` section of that constraint group.

You can define layers that are not valid layers in a `LEFDefaultRouteSpec` constraint group to be masterslice layers by assigning them the proper functions with techSetLayerFunction or techSetLayerFunctions. For more information about the `functions` subsection of the technology file, see functions in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_msLayers	A list of the valid masterslice layers contained in the specified technology database. The list has the following syntax: `list ( lt_mastersliceLayer … )` where, *lt_mastersliceLayer* is a valid masterslice layer.
`nil`	The technology database contains no valid masterslice layers or does not exist.

Example

```
techGetPrMastersliceLayers(tfID)
=> ("pdiff" "poly")
```

Returns a list of the valid masterslice layers, `pdiff` and `poly`, contained in the technology database identified by `tfID`.

techIsPrMastersliceLayer

```
techIsPrMastersliceLayer(
     d_techID
     tx_msLayer
     )
     => t / nil
```

Description

Verifies whether the specified layer in the specified technology database is a valid masterslice layer. A masterslice layer is any layer that meets both of the following conditions:

1. The layer is explicitly assigned one of the following functions: `nwell`, `pwell`, `ndiff`, `pdiff`, or `poly`. ASCII technology file location: in the `functions` subsection of the `layerRules` section.

2. If the technology database contains a `LEFDefaultRouteSpec` constraint group, the layer is not listed in the `validLayers` section of that constraint group.

You can define a layer that is not a valid layer in a `LEFDefaultRouteSpec` constraint group to be a masterslice layer by assigning it the proper function with `techSetLayerFunction`. For more information about the `functions` subsection of the technology file, see functions in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
tx_msLayer	The layer to check. Valid values: The layer name, the layer number

Value Returned

`t`	The specified layer is a valid masterslice layer in the specified technology database.
`nil`	The specified layer is not a valid masterslice layer or the technology database does not exist.

Example

```
techIsPrMastersliceLayer(tfID "diff")
=> t
```

Verifies that the layer `diff` is a valid masterslice layer in the technology database identified by `tfID`.

techSetPrViaRule

```
techSetPrViaRule(
     d_techID
     t_viaSpecID
     l_viaDefNames
     tx_layer1
     t_direction1
     l_params1
     tx_layer2
     t_direction2
     l_params2
     )
     => t / nil
```

Description

Updates the specified via specification with the specified via definitions in the technology database. ASCII technology file location: `viaSpecs` section. If the `viaSpecs` section does not exist, this function creates it.

For more information about the `viaSpecs` section of the technology file, see viaSpecs in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
t_viaSpecID	The database identifier of the via specification.
l_viaDefNames	A list of the name of each via definition (the *viaDefName* specified in the `viaDefs` section). The list has the following syntax: `list ( t_viaDefName … )` where, *t_viaDefName* is the name of a via definition to include in the via specification.
tx_layer1	The routing layer for the bottom of the via. Valid values: The layer name, the layer number
t_direction1	Ignored. The direction is taken from `routingDirections`.

l_params1

A list of parameters for the bottom routing layer. The list has the following syntax:

`list ( n_minWidth1 n_maxWidth1 n_overhang1 n_metalOverhang1 )`

where,

- *n_minWidth1* is the minimum width, in user units, of the bottom routing layer.
- *n_maxWidth1* is the maximum width, in user units, of the bottom routing layer.
- *n_overhang1* is the minimum spacing between the contact cut and the outer edge of the via.

  Valid values: When there is a single standard via definition in the via specification, *overhang1* must be equal to the *layer1Enc width* parameter in the `standardViaDef` subclass when the layer direction is `horizontal` or it must be equal to the *height* parameter when the layer direction is `vertical`. When there is no standard via definition or when there are multiple standard via definitions, *overhang1* must be `_NA_`.
- *n_metalOverhang1* is ignored if a value is specified. Valid Values: `_NA_`

tx_layer2

The routing layer for the top of the via.

Valid values: The layer name, the layer number

t_direction2

Ignored. The direction is taken from the `routingDirections`.

l_params2

A list of parameters for the top routing layer. The syntax is the same as for *l_params1*.

Value Returned

`t`	The rule was updated or added to the `viaSpecs` class in the specified technology database.
`nil`	The update was not done because one or both of the layers specified are not routing layers, the via definition does not exist, or the technology database does not exist.

Example

```
techSetPrViaRule(techID "viaSP21" (M2_M1)
"metal1" "vertical"   (.6  1.8   _NA_      _NA_)
"metal2" "horizontal" (.6  1.8   _NA_      _NA_)))
=> t
```

Appends the specified via definition to the `viaSpecs` class in the technology database identified by `techID`.

techGetPrViaRules

```
techGetPrViaRules(
     d_techID
     )
     => l_viaRules / nil
```

Description

Returns a list of the via specifications defined in the technology database. ASCII technology file location: `viaSpecs` section.

For more information about the `viaSpecs` class of the technology file, see viaSpecs in the *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_viaRules	A list of the names of the via specifications. The list has the following syntax: `( t_viaSpecName1 t_viaSpecName2 … )` where, *t_viaSpecName* is he name of the via specification (the *viaSpecName* specified in the `viaSpecs` class).
`nil`	The `viaSpecs` class does not exist, via definitions (`viaDefs`) contained in the via specifications specify layers that are not valid routing layers, or the technology database does not exist.

Example

```
techGetPrViaRules(techID)
=> ("viaSP21" "viaSP32")
```

Returns the names of the rules defined in the `viaSpecs` class in the technology database identified by `techID`.

techGetPrViaParams

```
techGetPrViaParams(
     d_techID
     t_viaSpecID
     )
     => l_viaParams / nil
```

Description

Returns the parameters assigned to the specified via specification in the technology database. ASCII technology file location: `viaSpecs` section.

For more information about the `viaSpecs` class of the technology file, see viaSpecs in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
t_viaSpecID	The database identifier of the via specification.

Value Returned

l_viaParams	A list of the parameters defined in the `viaSpecs` class for the specified via specification in the specified technology database. The list has the following syntax: `( lt_layer1 t_dir ( n_minWidth1 n_maxWidth1 n_overhang1 n_metalOverHang1 ) lt_layer2 t_dir2 ( n_minWidth2 n_maxWidth2 n_overhang2 n_metalOverHang2 ) )` where, ■ *lt_layer1* is the bottom routing layer of the via. ■ *t_dir1* is the preferred routing direction of the bottom routing layer as specified in the `layerRoutingGrids` subclass of the Physical Rules class. ■ *n_minWidth1* is the minimum width of the bottom routing layer, in user units.

- *n_maxWidth1* is the maximum width of the bottom routing layer, in user units.
- *n_overhang1* is the minimum spacing between the contact cut and the outer edge of the via.

  Note: *overhang1* is always _NA_ unless there is only one standard via definition, in which case, it is equal to the *layer1Enc width* parameter in the standardViaDef subclass when the layer direction is horizontal or it is equal to the *height* parameter when the layer direction is vertical.
- *n_metalOverhang1* is always _NA_.
- *lt_layer2* is the top routing layer of the via.
- *t_dir2* is the preferred routing direction of the top routing layer as specified in the layerRoutingGrids subclass of the Physical Rules class.
- *n_minWidth2* is the minimum width of the top routing layer, in user units.
- *n_maxWidth2* is the maximum width of the bottom routing layer, in user units.
- *n_overhang2* is the minimum spacing between the contact cut and the outer edge of the via.

  Note: *overhang2* is always _NA_ unless there is only one standard via definition, in which case, it is equal to the *layer2Enc width* parameter in the standardViaDef subclass when the layer direction is horizontal or it is equal to the *height* parameter when the layer direction is vertical.
- *n_metalOverhang2* is always _NA_.

nil — The technology database does not exist, the viaSpecs class does not exist, or the specified via specification is not defined in the viaSpecs class.

Example

```
techGetPrViaParams(techID "viaSP21")
=> ((M2_M1)
```

```
"metal1" "vertical"   (.6   1.8   _NA_   _NA_)
"metal2" "horizontal" (.6   1.8   _NA_   _NA_)))
```

Returns the specified via specification data from the `viaSpecs` section of the technology database identified by `techID`.

techSetPrGenViaRule

```
techSetPrGenViaRule(
     d_techID
     t_viaSpecName
     l_genViaRule
     )
     => t / nil
```

Description

Updates the specified via specification and any related standard via definitions in the specified technology database.

For more information about the `viaSpecs` section of the technology file, see viaSpecs in *Virtuoso Technology Data ASCII Files Reference*. For more information about the `viaDefs` section of the technology file and standard via definitions, see viaDefs in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
t_viaSpecName	The name of the via specification (the *viaSpecName* specified in the `viaSpecs` class).

l_genViaRule

A list containing the via specification parameters. The list has the following syntax:

```
t_cutLayer ( g_lowerPt g_upperPt g_xPitch
g_yPitch g_resistance ) tx_layer1 t_dir1
( n_minWidth1 n_maxWidth1 n_overhang1
n_metalOverHang1 ) tx_layer2 t_dir2
( n_minWidth2 n_maxWidth2 n_overhang2
n_metalOverHang2 )
```

- *t_cutLayer* is the cut layer specified in the standard via definition.

  Valid values: The layer name, the layer number

- *g_lowerPt* is the lower-left point of the left bottom cut.

  Valid values: Any x,y coordinate

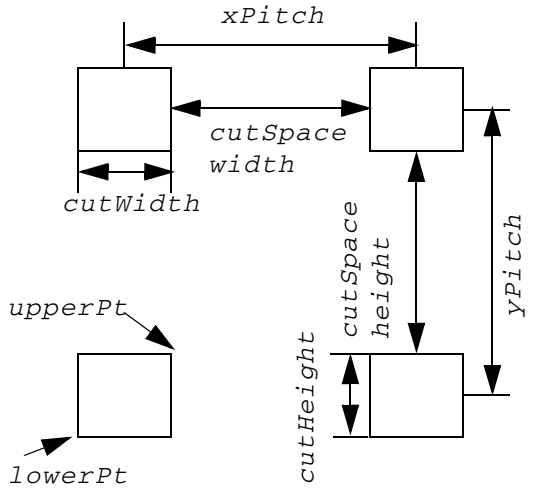
- *g_upperPt* is the upper-right point of the left bottom cut.

  Valid values: Any x,y coordinate

- *g_xPitch* is the distance between the centers of cuts in the x direction (`cutSpacing` in the x direction + `cutWidth`).

- *g_yPitch* is the distance between the centers of cuts in the y direction (`cutSpacing` in the y direction + `cutHeight`).

  The following figure illustrates *lowerPt*, *upperPt*, *xPitch*, and *yPitch*. These values are not mapped into the technology database or stored in the technology database; rather, they are computed when needed.



- *n_resistance* is *resistancePerCut* as specified in the standard via definition.
- *tx_layer1* is the bottom routing layer for the via as specified in the standard via definition.

  Valid values: The layer name, the layer number *t_dir1* is ignored. The direction is taken from the `layerRoutingGrids` subclass of the `layerRules` class.
- *n_minWidth1* is the minimum width, in user units, of the top layer.
- *n_maxWidth1* is the maximum width, in user units, of the top layer.

- *n_overhang1* is the minimum spacing between the contact cut and the outer edge of the via.

  Valid values: When there is a single standard via definition in the via specification, *overhang1* must be equal to the *layer1Enc width* parameter in the standardViaDef subclass when the layer direction is horizontal or it must be equal to the *height* parameter when the layer direction is vertical. When there is no standard via definition or when there are multiple standard via definitions, *overhang1* must be _NA_.

- *n_metal0Overhang1* is ignored if a value is specified.

  Valid values: _NA_

- *tx_layer2* is the top routing layer for the via as specified in the standard via definition.

  Valid values: the layer name, the layer number *t_dir2* is ignored. The direction is taken from the layerRoutingGrids subclass of the layerRules class.

- *n_minWidth2* is the minimum width, in user units, of the top layer.

- *n_maxWidth2* is the maximum width, in user units, of the top layer.

- *n_overhang2* is the minimum spacing between the contact cut and the outer edge of the via.

  Valid values: When there is a single standard via definition in the via specification, *overhang2* must be equal to the *layer2Enc width* parameter in the standardViaDef subclass when the layer direction is horizontal or it must be equal to the *height* parameter when the layer direction is vertical. When there is no standard via definition or when there are multiple standard via definitions, *overhang2* must be _NA_.

- *n_metal0Overhang2* is ignored if a value is specified. Valid Values: _NA_

Value Returned

`t`	The via specification and matching standard via definitions were updated or added to the `viaSpecs` and `viaDefs` classes in the specified technology database.
`nil`	The update was not done because one or both of the layers specified are not routing layers or the technology database does not exist.

Example

```
techSetPrGenViaRule(techID (viaspec9
via       (.6 .6  1.2    1.2    _NA_)
metal1   "horizontal" (.6  20.0  _NA_ _NA_)
metal2   "vertical"   (.6  20.0  _NA_ _NA_))
=> t
```

Updates the via specification `viaspec9` in the `viaSpecs` class of the technology database identified by `techID`.

techGetPrGenViaRules

```
techGetPrGenViaRules(
     d_techID
     )
     => l_genViaRules / nil
```

Description

Returns a list of all via specifications containing standard via definitions in the specified technology database.

For more information about the `viaSpecs` section of the technology file, viaSpecs in *Virtuoso Technology Data ASCII Files Reference*. For more information about the `viaDefs` section of the technology file and standard via definitions, see viaDefs in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_genViaRules	A list of the names of the via specifications. The list has the following syntax: `( t_viaSpecName1 t_viaSpecName2 … )` where, *t_viaSpecName* is the name of the via specification.
`nil`	The technology database does not exist or does not contain any standard via specifications.

Example

```
techGetPrGenViaRules(techID)
=> (viagen21 viagen32)
```

Returns a list of the names of the via specifications using standard via definitions in the technology database identified by `techID`.

techGetPrGenViaParams

```
techGetPrGenViaParams(
     d_techID
     t_viaSpecName
     )
     => l_genViaParams / nil
```

Description

Returns the parameters of all via specifications containing standard via definitions in the specified technology database. ASCII technology file location: `viaSpecs` section. Standard via definitions are defined in the `standardViaDefs` subsection of the `viaDefs` section.

For more information about the `viaSpecs` class of the technology file, see viaSpecs in *Virtuoso Technology Data ASCII Files Reference*. For more information about the `viaDefs` class of the technology file and standard via definitions, see viaDefs in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
t_viaSpecName	The name of the via specification.

Value Returned

l_genViaParams A list containing the data defined for the specified via specification in the `viaSpecs` class and the `standardViaDefs` subclass of the `viaDefs` class in the specified technology database. The list has the following syntax:

```
( lt_cutLayer ( g_lowerPt g_upperPt g_xPitch
g_yPitch n_resistance )
tx_layer1 t_dir1 ( n_minWidth n_maxWidth
n_overhang n_metalOverHang ) lt_layer2
t_dir2 ( n_minWidth n_maxWidth n_overhang
n_metalOverHang ) )
```

where,

- *tx_cutLayer* is the name of the cut layer as specified in the standard via definition.
- *g_lowerPt* is the lower left point of the left bottom cut.
- *g_upperPt* is the upper right point of the left bottom cut.
- *n_xPitch* is the distance between the centers of cuts in the x direction (`cutSpacing` in the x direction + `cutWidth`).
- *n_yPitch* is the distance between the centers of cuts in the y direction (`cutSpacing` in the y direction + `cutHeight`).
- *n_resistance* is *resistancePerCut* as specified in the standard via definition.
- *tx_layer1* is the bottom routing layer for the via as specified in the standard via definition.
- *t_dir1* is the preferred routing direction of the bottom layer.
- *n_minWidth1* is the minimum width, in user units, of the bottom layer.
- *n_maxWidth1* is the maximum width, in user units, of the bottom layer.

- *n_overhang1* is the minimum spacing between the contact cut and the outer edge of the via.

  Note: *overhang1* is always _NA_ unless there is only one standard via definition, in which case, it is equal to the *layer1Enc width* parameter in the standardViaDef subclass when the layer direction is horizontal or it is equal to the *height* parameter when the layer direction is vertical.

- *n_metal1Overhang1* is always _NA_. *tx_layer2* is the top routing layer for the via as specified in the standard via definition.

- *t_dir2* is the preferred routing direction of the top layer.

- *n_minWidth2* is the minimum width, in user units, of the top layer.

- *n_maxWidth2* is the maximum width, in user units, of the top layer.

- *n_overhang2* is the minimum spacing between the contact cut and the outer edge of the via.

  Note: *overhang2* is always _NA_ unless there is only one standard via definition, in which case, it is equal to the *layer2Enc width* parameter in the standardViaDef subclass when the layer direction is horizontal or it is equal to the *height* parameter when the layer direction is vertical.

nil — The technology database does not exist or the specified rule is not defined in the subclass.

Example

```
techGetPrGenViaParams(techID "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 all via specifications containing standard via definitions in the technology database identified by techID.

techSetPrNonDefaultRule

```
techSetPrNonDefaultRule(
     d_techID
     t_constraintGroupName
     l_layerConstsProps
     l_viaDefNames
     [ l_vias ]
     )
     => t / nil
```

Description

Creates or updates the named constraint group with the specified place and route constraints specified technology database. ASCII technology file location: `techLayerProperties` section.

Arguments

d_techID The database identifier of the technology database.

t_constraintGroupName

The name of the constraint group to create or update.

Valid values: Any unique string

l_layerConstsProps

A list of lists containing the constraints and layer properties. The list has the following syntax:

```
list ( list( tx_layer g_width g_space
[ n_notch ] g_wireExt g_cap g_resistance
[ g_edgecap ] ) … )
```

where,

- *tx_layer* is the routing layer, stored in the named constraint group as follows:

  ```
  interconnect( ( validLayers (layer) ) )
  ```

  Valid values: The layer name or the layer number

- *g_width* is the width, in user units, of a horizontal or vertical route segment on the specified layer, stored in the named constraint group as follows:

  ```
  spacings( ( minWidth layer value) )
  ```

- *g_space* is the space, in user units, between horizontal or vertical route segments on the specified layer, stored in the named constraint group as follows:

  ```
  spacings( ( minSpacing layer value) )
  ```

- *n_notch* is the notch spacing, in user units, allowed for the layer, stored in the named constraint group as follows:

  ```
  spacings( ( minSameNetSpacing layer value) )
  ```

- *g_wireExt* is the wire extension, in user units, for the layer, stored in the named constraint group as follows:

  ```
  spacings( ( minWireExtension layer value) )
  ```

- *g_cap* is the capacitance for the layer, stored in `techLayerProperties` as `areaCapacitance` for the layer.

- *g_resistance* is the resistance for the layer, stored in `techLayerProperties` as `sheetResistance` for the layer.

- *g_edgecap* is the edge capacitance for the layer, stored in `techLayerProperties` as `edgeCapacitance` for the layer.

l_viaDefNames

A list of via definition names. The list has the following syntax:

```
list ( t_viaDefName … )
```

where, *t_viaDefName* is the via definition name, stored in the named constraint group as follows:

```
interconnect( validVias ( viaDefName ) )
```

l_vias

A list of lists of via data. The list has the following syntax:

```
[ list( list( tx_viaLayer1 tx_viaLayer2
n_minSpace g_stack ) … ) ]
```

where,

- *tx_viaLayer1* specifies the bottom routing layer for the via.

  Valid values: The layer name, the layer number

- *tx_viaLayer2* specifies the top routing layer for the via.

  Valid values: The layer name, the layer number

- *n_minSpace* is the minimum spacing, in user units, allowed between via layers.

- *g_stack* indicates whether you can stack the via layers.

  Valid values: `t` (can be stacked), `nil` (cannot be stacked).

  This via data is stored in the named constraint group as follows:

```
spacings(
  ( minSameNetSpacing layer1 layer2 value )
  ( stackable layer1 layer2 t | nil )
)
```

Value Returned

`t`	The constraint group was created or updated, along with the specified layer properties.
`nil`	The technology database does not exist.

Example

```
techSetPrNonDefaultRule(tfID "rule1" list(
                                      list("metal1" 1.0 .3 .2 .1 20 .01 10)
                                    )
                                  list("M2_M1")
                                  list(
                                       list("metal1" "metal2" .5 t)
                                    )
                        )
=> t
```

Creates or edits the constraint group `rule1` in the technology database identified by `tfID`. The constraint group is as follows:

```
;( group     [override] )
;( -----     ---------- )
 ( "rule1"   nil
    interconnect(
     ( validLayers   (metal1  ) )
     ( validVias     (M2_M1_via  ) )
    ) ;interconnect
    spacings(
     ( minWidth                  "metal1"  1 )
     ( minSpacing                "metal1"  0.3 )
     ( minSameNetSpacing         "metal1"  0.2 )
     ( minWireExtension          "metal1"  0.1 )
     ( minSameNetSpacing         "metal1"  "metal2"  0.5 )
     ( stackable                 "metal1"  "metal2"  t )
    ) ;spacings
 ) ;rule1
```

Also adds the following to the `techLayerProperties` subsection of the `layerDefinitions` section of the technology database:

```
  ( areaCapacitanc          metal1          20.000000 )
  ( edgeCapacitance         metal1          10.000000 )
  ( sheetResistance         metal1          0.010000 )
```

techGetPrNonDefaultRules

```
techGetPrNonDefaultRules(
     d_techID
     )
     => l_nonDefaultRules / nil
```

Description

Returns a list of the place and route constraint groups in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_nonDefaultRules	A list of the rules. The list has the following syntax: `( t_rulename1 t_rulename2 … )`
nil	The technology database does not exist or does not contain the rules.

Example

```
techGetPrNonDefaultRules(techID)
=> ("NDrule1" "NDrule2")
```

techGetPrNonDefaultParams

```
techGetPrNonDefaultParams(
     d_techID
     t_name
     )
     => l_nonDefaultParams / nil
```

Description

Returns nondefault rules parameters from the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
t_name	The name of the rule.

Value Returned

l_nonDefaultParams	A list containing the data defined for the specified rule in the specified technology database.
nil	The technology database does not exist or does not contain the rules.

techSetPrRoutingPitch

```
techSetPrRoutingPitch(
     d_techID
     tx_layer
     n_pitch
     )
     => t / nil
```

Description

Updates the routing pitch of the specified layer in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The layer to update. Valid values: The layer name, the layer number
n_pitch	The routing pitch for the layer. Valid values: Any floating-point number, any integer

Value Returned

t	The data was updated in to the layerRoutingGrids subclass of the specified technology database.
nil	The layer is not in the LEFDefaultRouteSpec constraints group or the technology database does not exist.

Example

```
techSetPrRoutingPitch(techID "metal1" 2.4)
=> t
```

Updates the layerRoutingGrids subclass with the specified data in the technology database identified by techID.

techGetPrRoutingPitch

```
techGetPrRoutingPitch(
     d_techID
     tx_layer
     )
     => n_pitch / nil
```

Description

Returns the routing pitch defined for the specified layer in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The layer for which to get the routing pitch. Valid values: The layer name, the layer number

Value Returned

n_pitch	The routing pitch assigned to the layer in the `layerRoutingGrids` subclass; `0.0` (the default) if no routing pitch is assigned to the layer.
`nil`	The layer is not in the `LEFDefaultRouteSpec` constraint group or the technology database does not exist.

Example

```
techGetPrRoutingPitch(techID "metal1")
=> 2.4
```

Returns the routing pitch defined for the `metal1` layer in the `layerRoutingGrids` subclass in the Layer Rules class of the technology database identified by `techID`.

techSetPrRoutingOffset

```
techSetPrRoutingOffset(
     d_techID
     tx_layer
     n_offset
     )
     => t / nil
```

Description

Updates the routing offset of the specified layer in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The layer to update. Valid values: The layer name, the layer number
n_offset	The routing offset for the layer. Valid values: Any floating-point number, any integer

Value Returned

t	The data was updated in to the layerRoutingGrids subclass of the specified technology database.
nil	The layer is not in the LEFDefaultRouteSpec constraint group or the technology database does not exist.

Example

```
techSetPrRoutingOffset(techID "metal1" 0.2)
=> t
```

Updates the layerRoutingGrids subclass with the specified data in the technology database identified by techID.

techGetPrRoutingOffset

```
techGetPrRoutingOffset(
     d_techID
     tx_layer
     )
     => n_offset / nil
```

Description

Returns the routing offset defined for the specified layer in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The layer for which you want the routing offset. Valid values: The layer name, the layer number

Value Returned

n_offset	The routing offset assigned to the layer in the `layerRoutingGrids` subclass; `0.0` (the default) if no routing offset is assigned to the layer.
`nil`	The layer is not in the `LEFDefaultRouteSpec` constraint group or the technology database does not exist.

Example

```
techGetPrRoutingOffset(techID "metal1")
=> 0.2
```

Returns the routing offset defined for the `metal1` layer in the technology database identified by `techID`.

6

Database Constraints Functions

The database constraints functions are used to manipulate technology database constraints.

techIsDBUPerUUSet

```
techIsDBUPerUUSet(
     d_techID
     t_cvType
     )
     => t / nil
```

Description

Checks whether the number of database units per user unit for the specified cellview type in the specified technology database is set.

Arguments

d_techID	The database identifier of the technology database.
t_cvType	The name of the cellview type. Valid values: `maskLayout`, `schematic`, `netlist`, and `schematicSymbol`

Value Returned

`t`	The number of database units per user unit is set.
`nil`	The number of database units per user unit is not set.

Example

```
techIsDBUPerUUSet(tfID "maskLayout")
```

Sets the number of database units per user unit for the cellview type `maskLayout` in the technology database identified by `tfID`.

techSetDBUPerUU

```
techSetDBUPerUU(
     d_techID
     t_cvType
     n_dbuPerUU
     )
     => t / nil
```

Description

Sets the number of database units per each user unit for the specified cellview type in the specified technology database. **Caution:** Changing the number of database units per user unit with this function does not modify values previously stored in the database, which can result in misrepresented data. If you scale a full design or a cellview, you must use the `XScale` command in combination with modifying the technology database. For information about `XScale`, see *Virtuoso Studio Design Environment User Guide*.

Arguments

d_techID	The database identifier of the technology database.
t_cvType	The name of the cellview type. Valid values: `maskLayout`, `schematic`, `netlist`, and `schematicSymbol`
n_dbuPerUU	The number of database units per user unit. Valid values: Any numerical value

Value Returned

`t`	The number of database units per user unit is set.
`nil`	The technology database does not exist or the specified cellview type is invalid.

Example

```
techSetDBUPerUU(tfID "maskLayout" 1000.0)
```

Sets the number of database units per user unit for the cellview type `maskLayout` to `1000.0` in the technology database identified by `tfID`.

techGetDBUPerUU

```
techGetDBUPerUU(
     d_techID
     t_cvType
     )
     => f_dbuperuu / nil
```

Description

Returns the number of database units per user unit for the specified cellview type from the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
t_cvType	The name of the cellview type. Valid values: `maskLayout`, `schematic`, `netlist`, and `schematicSymbol`

Value Returned

f_dbuperuu	The number of database units per user unit.
`nil`	The technology database does not exist or the number of database units per user unit is not defined for the specified cellview type in the specified technology database.

Example

```
techGetDBUPerUU(tfID "maskLayout")
=> 1000.0
```

Returns the number of database units per user unit for the cellview type `maskLayout` in the technology database identified by `tfID`.

techIsUserUnitSet

```
techIsUserUnitSet(
     d_techID
     t_cvType
     )
     => t / nil
```

Description

Checks whether the user unit is set for the specified cellview type in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
t_cvType	The name of the cellview type. Valid values: `maskLayout`, `schematic`, `netlist`, and `schematicSymbol`

Value Returned

`t`	The user unit is set for the specified cellview type in the specified technology database.
`nil`	The user unit is not set.

Example

```
techIsUserUnitSet(tfID "maskLayout")
```

Checks if the user unit is set for the cellview type `maskLayout` in the technology database identified by `tfID`.

techSetUserUnit

```
techSetUserUnit(
     d_techID
     t_cvType
     t_userUnit
     )
     => t / nil
```

Description

Sets the user unit for the specified cellview type in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
t_cvType	The name of the cellview type. Valid values: `maskLayout`, `schematic`, `netlist`, and `schematicSymbol`
t_userUnit	The user unit. Valid values: `micron`, `millimeter`, `centimeter`, `meter`, `mil`, and `inch`

Value Returned

`t`	The user unit is set.
`nil`	The technology database does not exist or the specified cellview type is invalid.

Example

```
techSetUserUnit(tfID "maskLayout" "micron")
```

Sets the user unit for the cellview type `maskLayout` to `micron` in the technology database identified by `tfID`.

techGetUserUnit

```
techGetUserUnit(
     d_techID
     t_cvType
     )
     => t_userUnit / nil
```

Description

Returns the user unit that is set for the specified cellview type in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
t_cvType	The name of the cellview type. Valid values: `maskLayout`, `schematic`, `netlist`, and `schematicSymbol`

Value Returned

t_userUnit	The user unit that is set for the specified cellview type in the specified technology database.
`nil`	The technology database does not exist or the specified cellview type is invalid.

Example

```
techGetUserUnit(tfID "maskLayout")
=> "micron"
```

Returns the user unit `micron` for the cellview type `maskLayout` in the technology database identified by `tfID`.

7

Site Definitions Functions

The site definitions functions are used to create, access, and manipulate technology database site definitions.

techCreateScalarSiteDef

```
techCreateScalarSiteDef(
     d_techID
     t_siteDefName
     t_siteDefType
     n_width
     n_height
     [ g_symmetricInX = t | nil ]
     [ g_symmetricInY = t | nil ]
     [ g_symmetricInR90 = t | nil ]
     )
     => d_siteDefID / nil
```

Description

Creates in the specified technology database a definition of a site in which you can place cells in a row.

Arguments

d_techID	The database identifier of the technology database.
t_siteDefName	The name of the site definition to create. Valid values: Any string
t_siteDefType	The type of site definition to create. Valid values: `pad` and `core`
n_width	The width of the site, in user units. Valid values: Any number
n_height	The height of the site, in user units. Valid values: Any number
g_isSymmetricInX	Specifies whether the scalar site definition is symmetric in the X direction. Valid values: `t	nil` Default: `nil`

g_isSymmetricInY Specifies whether the scalar site definition is symmetric in the Y direction.

Valid values: `t | nil`

Default: `nil`

g_isSymmetricInR90

Specifies whether the scalar site definition is symmetric in rotation.

Valid values: `t | nil`

Default: `nil`

Value Returned

d_scalarSiteDefID

The site definition was created successfully with the database identifier *scalarSiteDefID*.

`nil` The technology database does not exist.

Example

```
techCreateScalarSiteDef(tfID "coreSite" "core" 574.84 1352.96 t nil t)
=> db:0x01d0600e
```

Creates a site definition named `coreSite` in the technology database identified by `tfID`. Its type is `core`, width is `574.84` user units, and height is `1352.96` user units, and it is symmetrical in the X direction and in rotation.

techCreateArraySiteDef

```
techCreateArraySiteDef(
     d_techID
     t_siteDefName
     t_siteDefType
     l_sitePattern
     [ g_symmetricInX = t | nil ]
     [ g_symmetricInY = t | nil ]
     [ g_symmetricInR90 = t | nil ]
     )
     => d_arraySiteDefID / nil
```

Description

Creates an array of scalar site definitions in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
t_siteDefName	The name of the array site definition to create. Valid values: Any string
t_siteDefType	The type of scalar site definitions to include in the array. Valid values: `pad` and `core`

l_sitePattern

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

```
list( list( t_siteName l_offset t_orient )… )
```

where,

- *t_siteName* is the name of the scalar site definition included in `arraySiteDef`.
- *l_offset* is a list specifying the X and Y offset for the scalar site definition. The list has the following syntax:

  ```
  list( g_xOffset g_yOffset )
  ```

- *t_orient* is the orientation for the scalar site definition.

  Valid values: `R0`, `R90`, `R180`, `R270`, `MX`, `MI`, `MXR90`, and `MYR90`

g_isSymmetricInX

Specifies whether the array site definition is symmetric in the X direction.

Valid values: `t | nil`

Default: `nil`

g_isSymmetricInY

Specifies whether the array site definition is symmetric in the Y direction.

Valid Values: `t | nil`

Default: `nil`

g_isSymmetricInR90

Specifies whether the array site definition is symmetric in rotation.

Valid Values: `t | nil`

Default: `nil`

Value Returned

d_arraySiteDefID	The site definition was created successfully with the database identifier *arraySiteDefID*.
nil	The technology database does not exist or one or more of the site definitions do not exist.

Example

```
techCreateArraySiteDef(
    tfID "myCoreArray" "core"
    list(list("coreSite" 0.0 0.0 "R270")
         list("coreSite2" 5.75 0.0 "MYR90")
        )
    )
=> db:0x01d0400e
```

Creates the array site definition `myCoreArray` in the technology database identified by `tfID`. The array site definition contains two scalar site definitions, `coreSite` (with X offset `0.0`, Y offset `0.0`, and orientation `R270`) and `coreSite2` (with X offset `5.75`, Y offset `0.0`, and orientation `MYR90`. None of the symmetric specifications is set, and so they all default to `nil`.

techFindSiteDefByName

```
techFindSiteDefByName(
     d_techID
     t_siteDefName
     )
     => d_siteDefID / nil
```

Description

Returns the database identifier of the named site definition from the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
t_siteDefName	The name of the site definition to find. Valid values: Any string

Value Returned

d_siteDefID	The database identifier of the site definition with the name *siteDefName*.
nil	The technology database does not exist or contains no site definition by the specified name.

Example

```
techFindSiteDefByName(tfID "siteDefA")
=> db:0x02b7a18c
```

Returns the database identifier for the site definition named `siteDefA` in the technology database identified by `tfID`.

techGetCellViewSiteDefs

```
techGetCellViewSiteDefs(
     d_cellviewID
     )
     => l_siteDefIDs / nil
```

Description

Returns a list of database identifiers of the site definitions used in the rows of the given cellview in the specified technology database.

Arguments

d_cellviewID	The database identifier for the cellview.

Value Returned

l_siteDefIDs	The database identifiers of the site definitions used in the cellview
nil	The technology database does not exist or no site definitions are used by the cellview.

Example

```
cvID = deGetCellView()
db:0x01d0800d
```

Returns the database identifier for the current cellview and assigns it to the variable `cvID`.

```
techGetCellViewSiteDefs( cvID )
(db:0x017b8f0c db:0x017b8f0d)
```

Returns the database identifiers for the site definitions for the current cellview.

techDeleteSiteDef

```
techDeleteSiteDef(
     d_siteDefID
     )
     => t / nil
```

Description

Deletes the specified site definition from the specified technology database.

Arguments

d_siteDefID	The database identifier of the site definition.

Value Returned

t	The specified site definition was successfully deleted.
nil	The specified database identifier does not exist.

Example

```
siteID = techFindSiteDefByName(tfID "siteDefA")
=> db:0x02b7a18c
```

Returns the database identifier for the site definition named `siteDefA` in the technology database identified by `tfID` and assigns it to the variable `siteID`.

```
techDeleteSiteDef(siteID)
=> t
```

Deletes the site definition identified by `siteID` from the current technology database.

8

Via Definitions and Via Specifications Functions

The via definition and specifications functions are used to manipulate technology database via definitions, via variants, and via specifications.

Via Definition SKILL Functions

The via definition functions are used to manipulate technology database via definitions.

cdsFixedViaDefs

```
cdsFixedViaDefs(
     l_list
     )
     => t / nil
```

Description

Specifies the technology class to define the cds fixed viaDef.

Arguments

l_list	List of layers.

Value Returned

t	The function successfully specifies the technology class to define the cds fixed viaDef.
nil	The function does not specify the technology class to define the cds fixed viaDef.

Example

```
cdsFixedViaDefs(
  (fixedVia1
    (layers
      (layer1   Metal1)
      (layer2   Metal2)
      (cutLayer Cont)
    )
  )
```

techCreateGenViaDef

```
techCreateGenViaDef(
     d_techID
     t_viaDefName
     tx_layer1 tx_layer2
     tx_cutLayer [l_extraLayers)(l_parameters)
     )
     => d_viaDefID / nil
```

Description

Creates a generated via definition in the specified technology database. ASCII technology file location: `cdsGenViaDefs` subsection in the `viaDefs` section.

For more information, see cdsGenViaDefs in the *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
t_viaDefName	The name of the generated via definition to create. Valid values: Any string unique in the database
tx_layer1	The bottom routing layer of the via. Valid values: The layer name or the layer number
tx_layer2	The top routing layer of the via. Valid Values: The layer name or the layer number
tx_cutLayer	The cut layer for the via definition.

l_extraLayers	A list of layers in this format:
	`'(extraLayers`
	`[ (layer1ExtraLayers` *l_layer1ExtraLayerNames*`)]`
	`[ (layer2ExtraLayers` *l_layer2ExtraLayerNames*`)]`
	`[ (cutExtraLayers` *l_cutExtraLayerNames*`)]`

l_parameters A list of parameters in this syntax:

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

Note: The `originOffset` parameter is valid only when `alignment` is set to `offset`.

Value Returned

d_viaDefID The generated via definition was created successfully with the database identifier *viaDefID*.

`nil` The generated via definition could not be created because either the technology database does not exist or there is no definition for the specified layer or layers in the technology database.

Example

```
techCreateGenViaDef(tech "myGenVia" "Metal1" "Metal2" "Via1"
'(extraLayers
      (layer1ExtraLayers ("PImplant" "Active"))
```

```
      (layer2ExtraLayers ("NImplant"))
      (cutExtraLayers ("Cut2"))
  )
  '(parameters
      (layer1Purpose "net")
      (layer2Purpose "net")
      (cutPurpose "fill")
      (cutWidth 0.1)
      (cutHeight 0.1)
      (cutColumns 6)
      (cutRows 8)
      (cutSpacing 0.1 0.1)
      (layer1Enc (0 0 0.1 0.1))
      (layer2Enc (0.2 0.2 0.2 0.2))
      (cutPattern ((0 1) (1 0)))
      (alignment "offset")
      (originOffset (0.1 0.1))
      (cutArraySpacing 0.1 0.2)
      (version 1)
      (layer1ExtraParams (
                     ((purpose "net") (enc (0.3 0.3 0.3 0.3)))
                     ((purpose "drawing") (enc (0.2 0.2 0.2 0.2)))
                   )
      )
      (layer2ExtraParams (((purpose "net") (enc (0.3 0.3 0.3 0.3)))))
      (cutArrayPatternX (2 4))
      (cutArrayPatternY (3 5))
  )
)
```

Creates a via between Metal1 and Metal2. For the Metal1 shape, additional shapes will be generated on PImplant and Active. For the Metal2 shape, an additional shape will be generated on NImplant. For each shape on Via1, an extra shape is generated on the Cut2 layer.

techCreateStdViaDef

```
techCreateStdViaDef(
     d_techID
     t_viaDefName
     t_layer1Name
     t_layer2Name
     l_cutLayerInfo
     l_cutArrayInfo
     ln_layer1Enc
     ln_layer2Enc
     ln_layer1Offset
     ln_layer2Offset
     ln_originOffset
     [ t_imp1 ln_imp1Enc
     [ t_imp2 ln_imp2Enc ] ]
     )
     => d_viaDefID / nil
```

Description

Creates a standard via definition in the specified technology database. This function will not create a standard via definition if the default parameters result in inverted shapes.

Arguments

d_techID	The database identifier of the technology database.
t_viaDefName	The name of the standard via definition to create.
	Valid values: Any string unique in the database
t_layer1Name	The name of the bottom routing layer of the via.
	Valid values: A string that is the name of a valid layer defined in the technology database
t_layer2Name	The name of the top routing layer of the via.
	Valid values: A string that is the name of a valid layer defined in the technology database

l_cutLayerInfo A list defining the cut layer. The list has the following syntax:

```
list ( t_cutLayerName n_cutWidth n_cutHeight
[ n_resPerCut ] )
```

- *t_cutLayerName* is the name of the cut layer.

  Valid values: A string that is the name of a valid layer with the layer function `cut` defined in the technology database

- *n_cutWidth* is the width, in user units, of a cut.

  Valid values: Any positive number

- *n_cutHeight* is the height, in user units, of a cut.

  Valid values: Any positive number

- *f_resPerCut* is the resistance per cut.

  Valid values: Any non-negative floating-point number

l_cutArrayInfo A list defining the cut array. The list has the following syntax:

```
list ( x_cutRows x_cutColumns ln_cutSpace )
```

- *x_cutRows* is the number of rows in the cut array.

  Valid values: Any positive integer

- *x_cutColumns* is the number of columns in the cut array.

  Valid values: Any positive integer

- *ln_cutSpace* is a list defining the distance between cuts and has the following syntax:

  ```
  list ( f_xCutSpacing f_yCutSpacing )
  ```

  - *f_xCutSpacing* is the horizontal distance, edge to edge, in user units, between cuts.

    Valid values: Any positive floating-point number

  - *f_yCutSpacing* is the vertical distance, edge to edge, in user units, between cuts.

    Valid values: Any positive floating-point number

ln_layer1Enc

A list defining the distance from the cut edge to the edge of layer 1. The list has the following syntax:

```
list ( f_layer1XEnc f_layer1YEnc )
```

- *f_layer1XEnc* is the horizontal distance, in user units, from the cut edge to the edge of layer 1.

  Valid values: Any floating-point number

- *f_layer1YEnc* is the vertical distance, in user units, from the cut edge to the edge of layer 1.

  Valid values: Any floating-point number

ln_layer2Enc

A list defining the distance from the cut edge to the edge of layer 2. The list has the following syntax:

```
list ( f_layer2XEnc f_layer2YEnc )
```

- *f_layer2XEnc* is the horizontal distance, in user units, from the cut edge to the edge of layer 2.

  Valid values: Any floating-point number

- *f_layer2YEnc* is the vertical distance, in user units, from the cut edge to the edge of layer 2.

  Valid values: Any floating-point number

ln_layer1Offset

A list defining the offset for layer 1. The list has the following syntax:

```
list ( f_layer1XOffset f_layer1YOffset )
```

- *f_layer1XOffset* is the horizontal offset, in user units.

  Valid values: Any floating-point number

- *f_layer1YOffset* is the vertical offset, in user units.

  Valid values: Any floating-point number

ln_layer2Offset A list defining the offset for layer 2. The list has the following syntax:

```
list ( f_layer2XOffset f_layer2YOffset )
```

- *f_layer2XOffset* is the horizontal offset, in user units.

  Valid values: Any floating-point number

- *f_layer2YOffset* is the vertical offset, in user units.

  Valid values: Any floating-point number

ln_originOffset A list defining the offset for the origin of the via. The list has the following syntax:

```
list ( f_originXOffset f_originYOffset )
```

- *f_originXOffset* is the horizontal offset for the origin of the via, in user units.

  Valid values: Any floating-point number

- *f_originYOffset* is the vertical offset for the origin of the via, in user units.

  Valid values: Any floating-point number

t_imp1 The implant layer 1 of the via.

Valid values: The layer name or the layer number

ln_imp1Enc A list defining the distance from the edge of layer 1 to the edge of implant layer 1. The list has the following syntax:

```
list ( f_imp1XEnc f_imp1YEnc )
```

- *f_imp1XEnc* is the horizontal distance, in user units, from the edge of layer 1 to the edge of implant layer 1.

  Valid values: Any floating-point number

- *f_imp1YEnc* is the vertical distance, in user units, from the edge of layer 1 to the edge of implant layer 1.

  Valid values: Any floating-point number

t_imp2 The implant layer 2 of the via.

Valid values: The layer name or the layer number

ln_imp2Enc

A list defining the distance from the edge of layer 2 to the edge of implant layer 2. The list has the following syntax:

```
list ( f_imp2XEnc f_imp2YEnc )
```

- *f_imp2XEnc* is the horizontal distance, in user units, from the edge of layer 2 to the edge of implant layer 2.

  Valid values: Any floating-point number

- *f_imp2YEnc* is the vertical distance, in user units, from the edge of layer 2 to the edge of implant layer 2.

  Valid values: Any floating-point number

Value Returned

d_viaDefID	The standard via definition was created successfully with the database identifier *viaDefID*.
nil	The technology database does not exist or there is no definition for a specified layer or layers in the technology database.

Example

```
myVia=techCreateStdViaDef(tfID "myVia" "M1" "M2"
list("VIA1" 0.5 0.5 5.0) list(2 2 '(0.15 0.15))
'(0.05 0.005) '(0.1 0.01) '(0.0 0.0) '(0.3 0.3) '(0.7 0.7) )

db:0x00ca233d
```

Creates the standard via definition named myVia in the technology database identified by tfID.

techCreateCustomViaDef

```
techCreateCustomViaDef(
     d_techID
     t_viaDefName
     d_cellviewID
     t_layer1Name
     t_layer2Name
     )
     => d_viaDefID / nil
```

Description

Creates a custom via definition in the specified technology database and associates it with the specified cellview.

Note: The function automatically sets the resistance per cut to `0.0`. You can reset it with techSetViaDefResistancePerCut.

Arguments

d_techID	The database identifier of the technology database.
t_viaDefName	The name of the custom via definition to create. Valid values: Any string unique in the database
d_cellviewID	The database identifier of the master cellview with which to associate the custom via definition. The cellview must be of type `via` and can be associated with only one custom via definition.
t_layer1Name	The name of the bottom layer of the via. Valid values: A string that is the name of a valid layer defined in the technology database
t_layer2Name	The name of the top layer of the via. Valid values: A string that is the name of a valid layer defined in the technology database

Value Returned

d_viaDefID	The custom via definition was created successfully with the database identifier *viaDefID*.
nil	The technology database does not exist or the specified cellview or one or both of the layers are not defined in the technology database.

Example

```
techCreateCustomViaDef(tfID "viaDef1" cvID "metal1" "metal2")
=> db:0x03b7a18c
```

Creates the via definition named viaDef1, which is associated with the cellview identified by cvID in the technology database identified by tfID, and returns its database identifier.

techCreateCustomViaDefByName

```
techCreateCustomViaDefByName(
     d_techID
     t_viaDefName
     t_libName
     t_cellName
     t_viewName
     t_layer1Name
     t_layer2Name
     )
     => d_viaDefID / nil
```

Description

Creates a custom via definition in the specified technology database and associates it with the master cellview identified by the specified library, cell, and view names.

Note: The function automatically sets the resistance per cut to `0.0`. You can reset it with techSetViaDefResistancePerCut.

Arguments

d_techID	The database identifier of the technology database.
t_viaDefName	The name of the custom via definition to create.
	Valid values: Any string unique in the database
t_libName	The name of the library containing the master cellview.
t_cellName	The cell name of the master cellview.
t_viewName	The view name of the master cellview.
t_layer1Name	The name of the bottom layer of the via.
	Valid values: A string that is the name of a valid layer defined in the technology database
t_layer2Name	The name of the top layer of the via.
	Valid values: A string that is the name of a valid layer defined in the technology database

Value Returned

d_viaDefID	The custom via definition was created successfully with the database identifier *viaDefID*.
nil	The technology database does not exist or the specified master cellview or one or both of the layers are not defined in the technology database.

Example

```
techCreateCustomViaDefByName(tfID "viaDef2" "lib1" "cell1" "symbolic" "metal1"
"metal2")
=> db:0x03b7a18c
```

Creates a via definition named viaDef2, which is associated with the master cellview in the library lib1, cell cell1, and view symbolic in the technology database identified by tfID, and returns its database identifier.

techSetViaDefResistancePerCut

```
techSetViaDefResistancePerCut(
     d_viaDefID
     f_resistancePerCut
     )
     )=> t / nil
```

Description

Sets the resistance per cut for the specified via definition in the current technology database.

Arguments

d_viaDefID	The database identifier of the via definition.
f_resistancePerCut	The resistance per cut to set for the via definition. Valid values: Any floating-point number

Value Returned

t	The resistance per cut is set for the via definition.
nil	The technology database does not exist or the specified via definition database identifier is not valid.

Example

```
vdID = techFindViaDefByName(tfID "viad")
=> db:0x01798a8c
```

Returns the database identifier for the via definition named `viad` in the technology database identified by `tfID` and assigns it to the variable `vdID`.

```
techSetViaDefResistancePerCut(vdID 1.4)
```

Sets the resistance per cut to `1.4` for the via definition identified by `vdID`.

techFindViaDefByName

```
techFindViaDefByName(
     d_techID
     t_viaDefName
     )
     => d_viaDefID / nil
```

Description

Returns the database identifier of the named via definition in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
t_viaDefName	The name of the via definition to find. Valid values: A string that is the name of a valid via definition

Value Returned

d_viaDefID	The database identifier of the via definition.
nil	The technology database does not exist or does not contain the named via definition.

Example

```
techFindViaDefByName(tfID "viaDef2")
=> db:0x03b7a18c
```

Returns the database identifier for the via definition named `viaDef2` in the technology database identified by `tfID`.

techDeleteViaDef

```
techDeleteViaDef(
     d_viaDefID
     )
     => t / nil
```

Description

Deletes the specified via definition from the current technology database.

Arguments

d_viaDefID	The database identifier of the via definition to delete.

Value Returned

t	The specified via definition was successfully deleted.
nil	The specified database identifier does not exist.

Example

```
vdID = techFindViaDefByName(tfID "viad")
=> db:0x01798a8c
```

Returns the database identifier for the via definition named `viad` in the technology database identified by `tfID` and assigns it to the variable `vdID`.

```
techDeleteViaDef(vdID)
=> t
```

Deletes the via definition identified by `vdID`.

Via Variant SKILL Functions

The via variant functions are used to manipulate technology database via variants.

techCreateGenViaVariant

```
techCreateGenViaVariant(
     d_techID
     t_viaVariantName
     t_viaDefName
     (l_parameters)
     )
     => d_viaVariantID / nil
```

Description

Creates a generated via variant matching the specified generated via definition in the specified technology database. ASCII technology file location: `cdsGenViaVariants` subsection in the `viaDefs` section.

For more information, see cdsGenViaVariants in the *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
t_viaVariantName	The name of the generated via variant definition to create. Valid values: Any string unique in the database
t_viaDefName	The name of the generated via definition to associate with the variant. See techCreateGenViaDef. Valid values: Any string unique in the database

l_parameters A list of parameters for the object in this syntax:

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

Note: The `originOffset` parameter is valid only when the `alignment` is set to `offset`.

Value Returned

d_viaVariantID The generated via variant was created successfully with the database identifier *viaVariantID*.

`nil` The generated via variant could not be created because either the technology database or the generated via definition does not exist.

Example

```
techCreateGenViaVariant(tech "vv1" "myGenVia"
  '(parameters
      (layer1Purpose "net")
      (layer2Purpose "net")
```

```
      (cutPurpose "fill")
      (cutWidth 0.1)
      (cutHeight 0.1)
      (cutColumns 6)
      (cutRows 8)
      (cutSpacing 0.1 0.1)
      (layer1Enc (0 0 0.1 0.1))
      (layer2Enc (0.2 0.2 0.2 0.2))
      (cutPattern ((0 1) (1 0)))
      (alignment "offset")
      (originOffset (0.1 0.1))
      (cutArraySpacing 0.1 0.2)
      (version 1)
      (layer2ExtraParams (((purpose "net") (enc (0.3 0.3 0.3 0.3)))))
      (layer1ExtraParams (
                     ((purpose "net") (enc (0.3 0.3 0.3 0.3)))
                     ((purpose "drawing") (enc (0.2 0.2 0.2 0.2)))
                   )
      )
      (cutLayerExtraParams (((purpose "net") (enc (0.3 0.3 0.3 0.3)))))
      (cutArrayPatternX (2 4))
      (cutArrayPatternY (3 5))
  )
)
```

Creates a generated via variant and associates it with the specified generated via definition.

techCreateStdViaVariant

```
techCreateStdViaVariant(
     (d_techfileId t_viaVariantName t_viaDefName)
     (t_cutLayerName f_cutLayerWidth f_cutLayerHeight)
     (n_cutRows n_cutCol (f_cutSpaceX f_cutSpaceY))
     (f_layer1EncX f_layer1EncY)
     (f_layer2EncX f_layer2EncY)
     (f_layer1OffsetX f_layer1OffsetY)
     (f_layer2OffsetX f_layer2OffsetY)
     (f_origOffsetX f_origOffsetY)
     (f_impant1EncX f_implant1EncY)
     (f_impant2EncX f_implant2EncY)
     (cut_pattern)
     )
     => d_viaVariantID / nil
```

Description

Creates a standard via variant in the current technology database. This function will not create a standard via variant by using the parameters that create inverted layer and implant enclosure shapes.

Arguments

d_techfileID	The technology file identifier of the technology database.
t_viaVariantName	The name of the standard via variant to create.
	Valid values: Any string unique in the database
t_viaDefName	The name of the standard via definition to create.
	Valid values: Any string unique in the database
t_cutLayerName	The name of the cut layer.
	Valid values: A string that is the name of a valid layer with the layer function `cut` defined in the technology database
t_cutLayerWidth	The width, in user units, of a cut.
	Valid values: Any positive number
t_cutLayerHeight	The height, in user units, of a cut.
	Valid values: Any positive number

n_cutRows	The number of rows in the cut array. Valid values: Any positive integer
n_cutCol	The number of columns in the cut array. Valid values: Any positive integer
f_cutSpaceX	The horizontal distance, edge to edge, in user units, between cuts. Valid values: Any positive floating-point number
f_cutSpaceY	The vertical distance, edge to edge, in user units, between cuts. Valid values: Any positive floating-point number
f_layer1EncX	The horizontal distance, in user units, from the cut edge to the edge of layer 1. Valid values: Any floating-point number
f_layer1EncY	The vertical distance, in user units, from the cut edge to the edge of layer 1. Valid values: Any floating-point number
f_layer2EncX	The horizontal distance, in user units, from the cut edge to the edge of layer 2. Valid values: Any floating-point number
f_layer2EncY	The vertical distance, in user units, from the cut edge to the edge of layer 2. Valid values: Any floating-point number
f_layer1OffsetX	The horizontal offset, in user units for layer 1. Valid values: Any floating-point number
f_layer1OffsetY	The vertical offset, in user units for layer 1. Valid values: Any floating-point number
f_layer2OffsetX	The horizontal offset, in user units for layer 2. Valid values: Any floating-point number
f_layer2OffsetY	The vertical offset, in user units for layer 2. Valid values: Any floating-point number

f_origOffsetX	The horizontal offset for the origin of the via, in user units. Valid values: Any floating-point number
f_origOffsetY	The vertical offset for the origin of the via, in user units. Valid values: Any floating-point number
f_impant1EncX	The horizontal distance, in user units, from the edge of layer 1 to the edge of implant layer 1. Valid values: Any floating-point number
f_impant1EncY	The vertical distance, in user units, from the edge of layer 1 to the edge of implant layer 1. Valid values: Any floating-point number
f_impant2EncX	The horizontal distance, in user units, from the edge of layer 2 to the edge of implant layer 2. Valid values: Any floating-point number
f_impant2EncY	The vertical distance, in user units, from the edge of layer 2 to the edge of implant layer 2. Valid values: Any floating-point number
cut_pattern	The cut pattern for the `viaVariant`. It is a 2-D array with the value of `0` or `1` for each entry. The value 0 represents no cut for the correspondent cut in the `viaVariant`. Whereas, the value 1 represents cut for the correspondent cut in the `viaVariant`.

Value Returned

d_viaVariantID	The technology file was created successfully with the database identifier *viaVariantID*.
`nil`	It fails to create `stdViaVariant` probably because of either the technology database does not exist or there is no definition for a specified layer(s) in the technology database.

Example

```
v2 = techCreateStdViaVariant(tf "via11" "M1_P" list("cont" 0.5 0.5) list(2 4
list(0.6 0.6) ) list(0.6 0.6) list(0.6 0.6) list(0.0 0.0) list(0.0 0.0) list(0.0
0.0) list(0.0 0.0) list(0.0 0.0) list(list(1 0 10) list(1 1 0 1)))

=> db:0x0601ea16
```

Returns the standard via variant in the current technology database.

techCreateCustomViaVariant

```
techCreateCustomViaVariant(
     d_techfileID
     t_viaVariantName
     t_viaDefName
     l_viaParams
     )
     => d_viaVariantID / nil
```

Description

Creates the custom via variant in the current technology database.

Arguments

d_techfileID	The technology file identifier of the technology database.
t_viaVariantName	The name of the standard via variant to create. Valid values: Any string that is unique in the database
t_viaDefName	The name of the standard via definition to create. Valid values: Any string that is unique in the database
l_viaParams	A list of the via parameters. Valid values: Any string unique in the database The list has the following syntax: *l_viaParams = ((t_paramName g_paramValue)…)* ■ *t_paramName* is the name of the via parameter. ■ *g_paramValue* is the value of the parameter.

Value Returned

l_viaVariantID	The technology file was created successfully with the database identifier *viaVariantID*.
nil	It fails to create customViaVariant probably because of either the technology database does not exist or there is no definition for a specified layer in the technology database.

Example

```
v3 = techCreateCustomViaVariant(tf "cvv" "myVia2" list(list("p1" 2) list("p2"
0.1)))
=> db:0x056de796
```

Returns the custom via variant in the current technology database.

techFindViaVariantByName

```
techFindViaVariantByName(
     d_techfileID
     t_viaVariantName
     )
     => d_viaVariantID / nil
```

Description

Finds the `viaVariantId` with the name *`t_viaVariantName`*.

Arguments

`d_techfileID`	The technology file identifier of the technology database.
`t_viaVariantName`	The name of the standard via variant that you attempt to find. Valid values: Any string unique in the database

Value Returned

`d_viaVariantID`	The technology file was created successfully with the database identifier *`viaVariantID`*.
`nil`	It fails to find `ViaVariant` probably because of either `d_techfileID` does not exist or the `viaVariant` with the name is not found in the technology database.

Example

```
techFindViaVariantByName(tf "via1")
=> db:0x056de792
```

Returns the `viaVariantId` with the name, `via1`.

Via Specifications SKILL Functions

The via specifications functions are used to manipulate technology database via specifications.

techCreateViaSpec

```
techCreateViaSpec(
     d_techID
     t_layer1
     t_layer2
     lt_defaultViaDefIDs
     )
     => d_viaSpecID / nil
```

Description

Creates a via specification with a default set of via definitions, as specified, in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
t_layer1	The first layer associated with the via specification. Valid values: Layer name or layer number
t_layer2	The second layer associated with the via specification. Valid values: Layer name or layer number
lt_defaultViaDefIDs	A list of the database identifiers of the default via definitions associated with the via specification. The list has the following syntax: `list ( t_viaDefID … )` **Note:** You can retrieve the database identifier for a via definition with techFindViaDefByName.

Value Returned

d_viaSpecID	The via specification was successfully created with the database identifier *viaSpecID*.
nil	The technology database does not exist or one or both of the layers or any of the via definitions are not in the technology database.

Example

```
techCreateViaSpec(tfID "Metal1" "Metal2" list("viaDef1" "viaDef2"))
=> db:0x03b7a18c
```

Creates a via specification with default via definitions `viaDef1` and `viaDef2` in the technology database identified by `tfID` and returns its database identifier.

techSetViaSpecTableEntry

```
techSetViaSpecTableEntry(
     d_viaSpecID
     n_layer1MinWidth
     n_layer1MaxWidth
     n_layer2MinWidth
     n_layer2MaxWidth
     l_viaDefIDs
     )
     => t / nil
```

Description

Sets the entry value of the via specification 2-D table with the specified width range values in the current technology database. If the width values specified are not already in the table, inserts them into the table in ascending order. The listed via definitions must exist.

Arguments

d_viaSpecID	The database identifier of the via specification.
n_layer1MinWidth	The minimum width, in user units, of the first layer associated with the via specification, if it is set. Valid values: Any number
n_layer1MaxWidth	The maximum width, in user units, of the first layer associated with the via specification, if it is set. Valid values: Any number
n_layer2MinWidth	The minimum width, in user units, of the second layer associated with the via specification, if it is set. Valid values: Any number
n_layer2MaxWidth	The maximum width, in user units, of the second layer associated with the via specification, if it is set. Valid values: Any number
l_viaDefIDs	A list of the database identifiers of the via definitions associated with the via specification to which the table entry applies. The list has the following syntax: `list ( t_viaDefID … )`

Value Returned

`t`	The via specification was successfully created with the database identifier *`viaSpecID`*.
`nil`	The technology database does not exist or one or both of the layers or any of the listed via definitions are not defined in the technology database.

Example

```
tfID=techGetTechFile(ddGetObj("newTech18"))
=> db:0x011ea00e
```

Returns the database identifier for the current technology database and assigns it to the variable `tfID`.

```
viaSpecID=techFindViaSpec(tfID 25 32)
=> db:0x02920592
```

Returns the database identifier for the via specification for layers `25` and `32`.

```
viadef1ID=techFindViaDefByName(tfID "viaDef1")
=> db:0x02920592
viadef2ID=techFindViaDefByName(tfID "viaDef2")
=> db:0x02920593
```

Returns the database identifiers for the via definitions, named `viaDef1` and `viaDef2`, in the via specification.

```
techSetViaSpecTableEntry(tfID viaSpecID 0.5 1.0 0.3 1.0 list(viadef1ID viadef2ID))
=> t
```

Sets the entry value of the via specification 2-D table to the values specified: `0.5` minimum width for layer 1, `1.0` maximum width for layer 1, `0.3` minimum width for layer 2, and `1.0` maximum width for layer 2 for the via definitions with the database identifiers `viadef1ID` and `viadef2ID`.

techGetViaSpecTableEntries

```
techGetViaSpecTableEntries(
     d_viaSpecID
     )
     => l_table / nil
```

Description

Returns the contents of the via specification, which is stored as a table, identified by the `viaSpec` database identifier from the current technology database.

Arguments

d_viaSpecID	The database identifier of the via specification.

Value Returned

l_table	A list of the table entries for the via specification's 2-D table of minimum and maximum widths for layer 1 and layer 2.The syntax is as follows: `( n_layer1MinWidth n_layer1MaxWidth n_layer2MinWidth n_layer2MaxWidthl lt_viaDefIDs )…` where, ■ *n_layer1MinWidth* is the minimum width, in user units, of the first layer associated with the via specification. ■ *n_layer1MaxWidth* is the maximum width, in user units, of the first layer associated with the via specification. ■ *n_layer2MinWidth* is the minimum width, in user units, of the second layer associated with the via specification. ■ *n_layer2MaxWidth* is the maximum width, in user units, of the second layer associated with the via specification. ■ *l_viaDefIDs* is a list of the database identifiers of the via definitions associated with the via specification to which the table entry applies. **Note:** Table entries with a value of `nil` are not returned.

`nil`	The technology database does not exist or the via specification does not exist.

Example

```
techGetViaSpecTableEntries( viaspecID )
=> ((0.12 1.1705 0.14 0.55
       (db:0x0258207d)
     )
     (0.12 1.1705 0.55 1.1705
       (db:0x02582117 db:0x025820b5 db:0x025820b4 db:0x0258207d)
     )
     (0.12 1.1705 1.1705 12.0005
       (db:0x0258207d)
     )
     (1.1705 12.0005 0.14 0.55
       (db:0x0258207d)
     )
     (1.1705 12.0005 0.55 1.1705
       (db:0x0258207d)
     )
     (1.1705 12.0005 1.1705 12.0005
       (db:0x0258207d)
     )
    )
```

Returns the table entries for the via specification identified by `viaSpecID` in the current technology database.

techGetViaSpecTableEntriesByName

```
techGetViaSpecTableEntriesByName(
     d_viaSpecID
     )
     => l_table / nil
```

Description

Generates all of the table entries in the via specification table for the specified via specification in the current technology database.

Arguments

d_viaSpecID	The database identifier of the via specification.

Value Returned

l_table	A list of the table entries for the via specification's 2-D table of minimum and maximum widths for layer 1 and layer 2, listed by name.
nil	The technology database does not exist or the via specification does not exist.

techGetViaSpecTableEntry

```
techGetViaSpecTableEntry(
     d_viaSpecID
     n_layer1Width
     n_layer2Width
     )
     => l_viaDefIDs / nil
```

Description

Returns a list of the viaDefIDs in the table entry with the specified layer widths in the requested via specification in the specified technology database.

Arguments

d_viaSpecID	The database identifier of the via specification.
n_layer1Width	The minimum width, in user units, of the first layer associated with the via specification table entry.
n_layer2Width	The minimum width, in user units, of the second layer associated with the via specification table entry.

Value Returned

l_viaDefIDs	A list of the database identifiers of the via definitions associated with the via specification to which the table entry applies. **Note:** Table entries with a value of `nil` are not returned.
`nil`	The technology database does not exist, the via specification does not exist, or no via definitions fitting the width specifications exist.

Example

```
techGetViaSpecTableEntry(viaSpecID 0.5 0.3)
=>(db:0x03b7a18c db:0x03b7a16d)
```

Returns the database identifiers for the via definitions in the table entry with minimum layer widths matching the specified widths, `0.5` for layer 1 and `0.3` for layer 2, in the via specification identified by `viaSpecID` in the current technology database.

techFindViaSpec

```
techFindViaSpec(
     d_techID
     tx_layer1Num
     tx_layer2Num
     )
     => d_viaSpecID / nil
```

Description

Returns the database identifier of any via specification containing the layers with the specified layer numbers in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
x_layer1Num	The first layer in the via specification. Valid values: Layer number
x_layer1Num	The second layer in the via specification. Valid values: Layer number

Value Returned

d_viaSpecID	The database identifier of the via specification.
nil	The technology database does not exist, one or more of the specified layers are not defined in the technology database, or the technology database does not contain a via specification for the specified layers.

Example

```
techFindViaSpec(tfID 30 34)
=> db:0x03b7a18c
```

Returns the database identifier for the via specification containing layers with numbers `30` and `34` in the technology database identified by `tfID`.

```
techFindViaSpec(tfID "Metal1" "Metal2")
=> db:0x03b7a18c
```

Returns the database identifier for the via specification containing layers with names `Metal1` and `Metal2`.

techDeleteViaSpec

```
techDeleteViaSpec(
     d_viaSpecID
     )
     => t / nil
```

Description

Deletes the specified via specification from the current technology database.

Arguments

d_viaSpecID	The database identifier of the via specification to delete.

Value Returned

t	The specified via specification was successfully deleted.
nil	The specified database identifier does not exist.

Example

```
viaSpecID = techFindViaSpec(tfID "Metal1" "Metal2")
=> db:0x03b7a18c
```

Returns the database identifier for the via specification named `VIA1ARRAY` in the technology database identified by `tfID` and assigns it to the variable `viaSpecID`.

```
techDeleteViaSpec(viaSpecID)
=> t
```

Deletes the via specification identified by `viaSpecID`.

9

Device Functions

The device functions are used to manipulate the data in the `devices` section of the technology database.

techGetDeviceCellView

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

Description

Loads the supermaster cellview of the specified device into virtual memory and returns the associated database identifier. The cellview opens in read mode. You can close the cellview with dbClose, as defined in *Virtuoso Studio Design Environment SKILL Reference*. The cellview must be defined as a device in the specified technology database. ASCII technology file location: `devices` section.

For more information about the `devices` section, see Technology File Devices in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
t_deviceName	The name of the device.
t_viewName	The view name of the device.

Value Returned

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

Example

```
techGetDeviceCellView(tfID "NMOS1" "layout")
=> db:20953132
```

Opens the `layout` view of the `NMOS1` cell and returns the database identifier `20953132`.

techGetDeviceCParam

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

Description

Returns a list of the names and values of the class parameters of the specified device from the specified technology database. ASCII technology file location: `devices` section

For more information about the `devices` section, see Technology File Devices in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
t_deviceName	The name of the device.
t_viewName	The view name of the device.

Value Returned

l_paramValue	A list of class parameter name and value pairs. The list has the following syntax: `( ( t_paramName g_paramValue )…)` where, ■ *t_paramName* is the name of the class parameter. ■ *g_paramValue* is the value assigned to the parameter when the device was created. `nil` indicates that the parameter takes the technology database default value.
`nil`	The technology database does not exist or the device is not defined.

Example

```
techGetDeviceCParam(tfID "NMOS1" "layout")
=> (("userFunc" "")
       ("abutClass" "abut1")
       ("diffusionWidth" nil)
       ("diffusionSpacing" nil)
       ("diffusionPolyEnclosure" nil)
       ("diffusionContactEnclosure" nil)
       ("yContactLayerL" nil)
       ("xContactLayerW" nil)
       ("yContactSpacing" nil)
       ("xContactSpacing" nil)
       ("metalContactEnclosure" nil)
       ("polyWidth" nil)
       ("polyPolySpacing" nil)
       ("polyDiffusionSpacing" nil)
       ("polyContactSpacing" 2.0)
       ("polyDiffusionExtension" nil)
       ("contactSpacingMethod" 1)
       ("stretchHandles"
            (t nil t nil nil
               t t nil
             )
       )
       ("drainTerminalName" "D")
       ("sourceTerminalName" "S")
       ("gateTerminalName" "G")
       ("implantLayers" nil)
       ("diffusionLayer" "pdiff")
       ("contactLayer" "cont")
       ("metalLayer" "metal1")
       ("gatePolyLayer" "poly1")
       ("classVersion" 1)
   )
```

Returns the class parameters defined for the device named `NMOS1` in the technology database identified by `tfID`. The parameters returned with a value of `nil` all take their respective technology database default value; the others take their specified values (for example, `polyContactSpacing` takes the value `2.0` rather than the `minSpacing` value specified in the technology database for the poly and contact layers.

techGetDeviceFParam

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

Description

Returns a list of the names and values of the formal parameters defined for the specified device from the specified technology database. ASCII technology file location: `devices` section.

For more information about the `devices` section, refer to "Technology File Devices" in the *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
t_deviceName	The name of the device.
t_viewName	The view name on which the device is defined.

Value Returned

l_paramValue	A list of formal parameter name and default pairs. The list has the following syntax: `( ( t_paramName g_paramValue )… )` where, ■ *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 database does not exist or the device is not defined.

Example

```
techGetDeviceFParam(techID "NMOS1" "layout")
=> (("formalVersion" 0)
       ("fingerWidth" 2.0)
       ("fingerLength" 1.0)
       ("numFinger" 1)
       ("leftAbutmentState" 1)
       ("rightAbutmentState" 1)
       ("contactList" nil)
       ("metalList" nil)
       ("gateExtStretch" nil)
       ("diffStretchLH" 0.0)
       ("diffStretchRH" 0.0)
       ("diffStretchLTV" 0.0)
       ("diffStretchLBV" 0.0)
       ("diffStretchRTV" 0.0)
       ("diffStretchRBV" 0.0)
       ("sourceFirst" "TRUE")
       ("metalContactEncTop" 0.0)
       ("metalContactEncBottom" 0.0)
       ("metalContactEncInner" 0.0)
       ("metalContactEncOuter" 0.0)
       ("metalContactEncOverride" "")
       ("userArgs" "")
)
```

Returns the formal parameters and values for the `NMOS1` device defined in the technology database identified by `tfID`.

techGetDeviceInClass

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

Description

Returns a list of the names of all devices of the specified device type and view name from the specified technology database. ASCII technology file location: `devices` section.

For more information about the `devices` section, see Technology File Devices in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
t_deviceType	The device type name.
t_viewName	The view name on which the device is defined.

Value Returned

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

Example

```
techGetDeviceInClass(tfID "guardring" "layout")
=> ("NMOS1" "NMOS2")
```

Returns the `NMOS1` and `NMOS2` device names, which are `guardring` devices defined in the technology database identified by `tfID`.

techGetDeviceClassViewList

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

Description

Returns a list of the view names used by devices from the specified technology database. ASCII technology file location: `devices` section.

For more information about the `devices` section, see Technology File Devices in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_deviceClassViews	A list of the view names used for devices in the technology database.
`nil`	The technology database does not exist, or no devices are defined.

Example

```
techGetDeviceClassViewList(tfID)
=> ("layout")
```

Returns the view name `layout` for the device classes in the technology database identified by `tfID`.

techRegisterUserDevice

```
techRegisterUserDevice(
     t_techLibName
     t_className
     t_viewName
     t_devName
     )
     => t / nil
```

Description

Appends device name to the list of devices returned by call of techGetDeviceInClass.

Arguments

t_techLibName	The design data identifier for the DFII library.
t_className	The class of a device.
t_viewName	The view name on which the device is defined.
t_devName	The name of a device.

Value Returned

t	A device name is appended to the list.
nil	The technology database does not exist, or no devices are defined.

Example

```
techRegisterUserDevice("lib" "gdsGuardRing" "layout" "grDev")
```

Appends device named grDev to the list of devices.

techUnregisterUserDevice

```
techUnregisterUserDevice(
     t_techLibName
     t_className
     t_viewName
     [t_devName]
     )
     => t
```

Description

Removes device name from the list of devices returned by techGetDeviceInClass. The device name is omitted then all the devices registered for given class/view/libname will be unregistered. Only devices added by techRegisterUserDevice gets affected.

Arguments

t_techLibName	The design data identifier for the DFII library.
t_className	The class of a device.
t_viewName	The view name on which the device is defined.
t_devName	The name of a device.

Value Returned

t	A device name gets removed from the list.

Example

```
techUnregisterUserDevice("lib" "gdsGuardRing" "layout")
```

Removes the device name from the list.

techGetDeviceTechFile

```
techGetDeviceTechFile(
     d_deviceID
     )
     => d_techID / nil
```

Description

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

Arguments

d_deviceID	The database identifier of the device.

Value Returned

d_techID	The database identifier of the technology database bound to the specified device.
nil	The device does not exist or its technology database cannot be determined.

Example

```
techGetDeviceTechFile (devID)
=> db:25675212
```

Returns the database identifier for the technology database bound to the device identified by devID.

techIsDevice

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

Description

Indicates whether the supermaster of the specified instance is defined as a device in the in the current technology database. ASCII technology file location: `devices` section.

For more information about the `devices` section, see Technology File Devices in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_instID	The database identifier of the instance.

Value Returned

`t`	The supermaster of the specified instance is a device.
`nil`	The technology database does not exist or the cellview is not defined as a device.

Example

```
instID=dbFindAnyInstByName(cvID "I0")
=> db:0x025c2694
```

Returns the database identifier for the instance `I0` and stores it in `instID`.

```
techIsDevice(instID)
=> t
```

Instantiates the instance identified by `instID` from a cellview defined as a device in the current technology database.

techSetDeviceProp

```
techSetDeviceProp(
     d_techID
     t_deviceName
     t_viewName
     l_property
     )
     => t / nil
```

Description

Sets or updates the value of the specified device property in the specified technology database. If the property does not exist, this function creates it. When you create a device or set a property on a device, the software adds the device name, view name, property name, and property value for each property on the device.

For more information about the Devices class, see Technology File Devices in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
t_deviceName	The device name.
t_viewName	The view name on which the device is defined. Valid values: `layout`
l_property	A list of the property name and value. The list has the following syntax: `( t_propName g_value )` where, ■ *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

Value Returned

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

Example

```
techSetDeviceProp(tfID "MOS1" "layout" list("prop1" 2.0))
```

Sets the property `prop1` to `2.0` for the `MOS1` device in the technology database identified by `tfID`.

techGetDeviceProp

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

Description

Returns the value of the specified device property from the specified technology database.

For more information about the Devices class, see Technology File Devices in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
t_deviceName	The device name.
t_viewName	The view name on which the device is defined. Valid values: `symbolic`
t_propName	The property name.

Value Returned

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

Example

```
techGetDeviceProp(tfID "MOS1" "layout" "prop1")
=> 2.0
```

Returns the value, `2.0`, of the property `prop1` defined for the `MOS1 layout` device in the technology database identified by `tfID`.

techGetDeviceClass

```
techGetDeviceClass(
     d_techID
     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 specified technology database. ASCII technology file location: `devices` section. If you specify a device name, this function returns the device type of that device.

For more information about the `devices` section, see Technology File Devices in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
t_deviceName	The device name. Valid values: Name of an existing device, `nil`
t_viewName	The view name defined for the device type.

Value Returned

l_types	A list of device type names of all device types defined with the specified view in the current technology database. 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 database or device does not exist.

Examples

```
techGetDeviceClass(tfID "MOS1" "layout")
=> "guardring"
```

Returns the device type, `guardring`, of the `MOS1 layout` device.

techGetInstDeviceClass

```
techGetInstDeviceClass(
     d_instID
     )
     => t_deviceClassName / nil
```

Description

Returns the device class of the specified instance in the current technology database. ASCII technology file location: `devices` section.

For more information about the `devices` section, see Technology File Devices in *Virtuoso Technology Data ASCII Files Reference*. For information about retrieving the instance database identifier, see `dbFindAnyInstByName` in *Virtuoso Studio Design Environment SKILL Reference*.

Arguments

d_instID	The database identifier of the instance.

Value Returned

t_deviceClass	The device class (type) of the instance.
nil	The technology database or device does not exist.

Examples

```
dbCreateInstByMasterName(cvID "cellTechLib" "MOS1" "layout" "inst2" list(0 0) "R0"
1)
```

Creates the instance `inst2` from the master `MOS1`.

```
instID=dbFindAnyInstByName(cvID "inst2")
=> db:41956120
```

Assigns the instance database identifier to the variable `instID`.

```
techGetInstDeviceClass(instID)
=> "guardring"
```

Returns the device class, `guardring`, of the instance `inst2`.

techSetDeviceClassProp

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

Description

Updates the value of the specified device type property in the specified technology database. ASCII technology file location: `devices` section. If the property does not exist, this function creates it.

For more information about the `devices` section, see Technology File Devices in the *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
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

Value Returned

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

Example

```
techSetDeviceClassProp(tfID " "leTran" "layout" list("function" "transistor"))
```

Sets the `function` property of the device type `leTran`, view name `layout` to `transistor` in the technology database identified by `tfID`.

techGetDeviceClassProp

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

Description

Returns the value of the specified device type property from the specified technology database.

For more information about the `devices` section, see Technology File Devices in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
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.

Value Returned

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 database does not exist, the device type property is not defined, or the property value is `nil`.

Example

```
techGetDeviceClassProp(tfID "guardring" "layout" "function")
=> "transistor"
```

Returns the value, `transistor`, of the `function` property on the `guardring` device type defined for the `layout` view in the technology database identified by `tfID`.

techDeleteDeviceClass

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

Description

Deletes all devices of the specified device class and view from the specified technology database.

For more information about the `devices` section, see Technology File Devices in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
t_viewName	The name of the view for the device(s) you want to delete.
t_className	The class of devices you want to delete.

Value Returned

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

Example

Suppose that a technology database contains the following data for creating a device:

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

; class parameter name value pairs
   ()

; formal parameter name value pairs
   (
```

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

; access techdb controls 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")
)



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

Deletes the device created with the technology data listed above.

techSetMPPTemplate

```
techSetMPPTemplate(
                (    d_techID
                     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]
            [l_rodConnectivityArgs for subrectangles]
            [n_beginSegmentOffset]
            [n_endSegmentOffset]

        ) ; end of first subrectangle list
        …
) ; end of all subrectangle lists

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

Description

Defines a 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_techID	The database identifier of the technology database.
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 SKILL Reference*.

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_fromObj` *`txf_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 SKILL Reference*.

In the `multipartPathTemplates` section of *Virtuoso Technology Data ASCII Files Reference*, 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_techID
     )
     => l_mppTemplateNames / nil
```

Description

Returns the names of all multipart path (MPP) templates defined in the current technology database.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_mppTemplateNames	A list of the names of the MPP templates defined in the technology database.
nil	The technology database does not exist or the does not contain MPP template definitions.

Example

```
techGetMPPTemplateNames(tfID)
=> ("ntran" "ntran1" "guardRingP" "guardRing")
```

Returns the names of the MPP templates defined in the technology database identified by `tfID`.

techGetMPPTemplateByName

```
techGetMPPTemplateByName(
     d_techID
     t_mppTemplateName
     )
     => l_template / nil
```

Description

Returns the definition of the specified multipart path (MPP) template in the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
t_mppTemplateName	The name of the MPP template to retrieve.

Value Returned

l_template	A list defining the MPP template. The elements of this list match the *l_template* arguments for techSetMPPTemplate.
nil	The technology database does not exist or the technology library does not contain the named MPP template definition.

Example

```
techGetMPPTemplateByName(tfID "guardRing")

=> ((("diff" "drawing") 4.0 nil nil nil
                nil nil nil
           ) nil
           ((("metal1" "drawing") 0.6 t nil nil
                   ("inputOutput" t
                        ("right" "left" "bottom") nil
                   )
               )
           )
           ((("cont" "drawing") 1.0 nil t nil
                   nil nil -1.2 nil nil
               )
           )
        )
```

Returns the definition of the MPP template named `guardRing` in the technology database identified by `tfID`.

techGetExtractDevices

```
techGetExtractDevices(
     d_techID
     )
     => l_extractDevices / nil
```

Description

Returns a list of all extract devices defined in the specified technology database. ASCII technology file location: `devices` section.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

l_extractDevices	A list of the extract devices.
nil	The technology database does not exist or does not contain extract devices.

techSetExtractMOS

```
techSetExtractMOS(
     d_techID
     l_deviceDefinition
     )
     => t / nil
```

Description

Appends the specified `extractMOS` specification to the specified technology database. ASCII technology file location: `extractMOS` subsection of the `devices` section. If an `extractMOS` device of the same name exists, this function replaces it with the new specification. If `the extractMOS` subsection does not exist, this function creates it.

Arguments

d_techID	The database identifier of the technology database.
l_deviceDefinition	A list defining the `extractMOS` specification. The list is in the format: `list ( t_deviceName tx_recognitionLayer tx_gateLayer tx_sourceDrainLayer tx_bulkLayer [ tx_modelName ] )`

Value Returned

`t`	The `extractMOS` device specification is successfully added to the technology database.
`nil`	The technology database does not exist or does not contain `extractMOS` devices.

techGetExtractMOS

```
techGetExtractMOS(
     d_techID
     tx_deviceName
     )
     => l_deviceDefinition / nil
```

Description

Returns the definition of the named `extractMOS` device from the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
tx_deviceName	The name of the device.

Value Returned

l_deviceDefinition

A list defining the `extractMOS` specification. The list is in the format:

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

`nil` The technology database does not exist or does not contain the named `extractMOS` device.

techSetExtractRES

```
techSetExtractRES(
     d_techID
     l_deviceDefinition
     )
     => t / nil
```

Description

Appends the specified `extractRES`specification to the specified technology database. ASCII technology file location: `extractRES` subsection of the `devices` section. If an `extractRES` device of the same name exists, this function replaces it with the new specification. If `the extractRES` subsection does not exist, this function creates it.

Arguments

d_techID	The database identifier of the technology database.
l_deviceDefinition	A list defining the `extractRES` specification. The list is in the format: `list ( t_deviceName tx_recognitionLayer tx_termLayer [ tx_modelName ] )`

Value Returned

`t`	The `extractRES` device specification is successfully added to the technology database.
`nil`	The technology database does not exist or does not contain `extractRES` devices.

techGetExtractRES

```
techGetExtractRES(
     d_techID
     tx_deviceName
     )
     => l_deviceDefinition / nil
```

Description

Returns the definition of the named `extractRES` device from the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
tx_deviceName	The name of the device.

Value Returned

l_deviceDefinition

A list defining the `extractRES` specification. The list is in the format:

```
( t_deviceName tx_recognitionLayer
tx_termLayer [ tx_modelName ] )
```

`nil` The technology database does not exist or does not contain the named `extractRES` device.

techSetExtractCAP

```
techSetExtractCAP(
     d_techID
     l_deviceDefinition
     )
     => t / nil
```

Description

Appends the specified `extractCAP` specification to the specified technology database. ASCII technology file location: `extractCAP` subsection of the `devices` section. If an `extractCAP` device of the same name exists, this function replaces it with the new specification. If `the extractCAP` subsection does not exist, this function creates it.

Arguments

d_techID	The database identifier of the technology database.
l_deviceDefinition	A list defining the `extractCAP` specification. The list is in the format: `list ( t_deviceName tx_recognitionLayer tx_plusLayer tx_minusLayer [ tx_modelName ] )`

Value Returned

`t`	The `extractCAP` device specification is successfully added to the technology database.
`nil`	The technology database does not exist or does not contain `extractCAP` devices.

techGetExtractCAP

```
techGetExtractCAP(
     d_techID
     tx_deviceName
     )
     => l_deviceDefinition / nil
```

Description

Returns the definition of the named `extractCAP` device from the technology database.

Arguments

d_techID	The database identifier of the technology database.
tx_deviceName	The name of the device.

Value returned

l_deviceDefinition

A list defining the `extractCAP` specification. The list is in the format:

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

`nil` The technology database does not exist or does not contain the named `extractCAP` device.

techSetExtractDIODE

```
techSetExtractDIODE(
     d_techID
     l_deviceDefinition
     )
     => t / nil
```

Description

Appends the specified `extractDIODE` specification to the specified technology database. ASCII technology file location: `extractDIODE` subsection of the `devices` section. If an `extractDIODE` device of the same name exists, this function replaces it with the new specification. If `the extractDIODE` subsection does not exist, this function creates it.

Arguments

d_techID	The database identifier of the technology database.
l_deviceDefinition	A list defining the `extractDIODE` specification. The list is in the format: `list ( t_deviceName tx_recognitionLayer tx_plusLayer tx_minusLayer [ tx_modelName ] )`

Value Returned

`t`	The `extractDIODE` device specification is successfully added to the technology database.
`nil`	The technology database does not exist or does not contain `extractDIODE` devices.

techGetExtractDIODE

```
techGetExtractDIODE(
     d_techID
     tx_deviceName
     )
     => l_deviceDefinition / nil
```

Description

Returns the definition of the named `extractDIODE` device from the specified technology database.

Arguments

d_techID	The database identifier of the technology database.
tx_deviceName	The name of the device.

Value Returned

l_deviceDefinition

A list defining the `extractDIODE` specification. The list is in the format:
`( t_deviceName tx_recognitionLayer tx_plusLayer tx_minusLayer [ tx_modelName ] )`

`nil` The technology database does not exist or does not contain the named `extractDIODE` device.

techCreateWaveguideDef

```
techCreateWaveguideDef(
     d_techFileId
     t_name (tx_layer tx_purpose)
     l_derivedShapeSpecs
     [?minWidth n_minWidth]
     [?minBendRadius n_minBendRadius]
     [?modeProperties l_modeProperties]
     [?maxTaperAngle f_maxTaperAngle]
     )
     => d_waveguideDefId / nil
```

Description

(Virtuoso Photonics Option) Creates a waveguideDef object in the specified technology file and returns the database ID of the object.

Arguments

d_techFileId	The database ID of the technology file in which the waveguideDef object is to be created.
t_name	The unique name of the new waveguideDef object.

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

l_derivedShapeSpecs	

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

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

Here:

- *`tx_layer`*: The layer of the derived shape.
- *`tx_purpose`*: The corresponding purpose for the layer. If not specified, the value is set as `drawing`.
- The type of the derived shape can be one of the following:
  - `enclosure`: Centers the shape around the master shape based on the *`tn_enclosure`* value specified. By default, the value is `nil`.
  - `offset`: Offsets the shape from the master shape. The specification includes one or more of these attributes: `side` (direction as `both`, `right`, or `left`), `inner` (offset of the inner edge of the derived shape from the centerline of the master shape), and `outer` (offset of the outer edge of the derived shape from the edge of the master shape). The default values are `both`, `nil`, and `nil`.

`?minWidth` *`n_minWidth`*

The minimum width of the object.

`?minBendRadius` *`n_minBendRadius`*

The minimum bend radius of the object.

l_modeProperties — The mode properties of the waveguide object as a list in the following format:

`list(`*t_functionName* `list(`*t_mode n_index*`)...)`

Here:

- *t_functionName*: The name of a SKILL function that returns the mode properties of the waveguide.
- *t_mode n_index*: One of a series of tuples representing modes. Each tuple must be unique in the series and must include a name (a string) and an index (an integer greater than zero).

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

Value Returned

d_waveguideDefId — The database ID of the new waveguideDef object.

`nil` — The waveguideDef object could not be created, possibly because the technology file does not exist or the specified LPP is invalid.

Example

```
wgDef = techCreateWaveguideDef(tf
    "wgDefName"
    list("wgLayer" "drawing")
    '(
        (("NWell" "drawing") enclosure 0.8)
        ("Active" offset side "both" inner 0.1 outer "outerEnvValue+10")
    )
    ?minWidth 0.6
    ?minBendRadius 0.2
    ?modeProperties '("modeFunction" (("mode1" 1) ("mode2" 2)))
)
```

Creates a waveguideDef object by the name `wgDefName` on the `wgLayer/drawing` LPP. The object has a minimum width of `0.6` and a minimum bend radius of `0.2`. The mode properties for the waveguide will be returned by the SKILL function named `modeFunction`. There are two modes, `mode1` and `mode2`, with indexes `1` and `2`.

It specifies the following shapes relative to the master shape on the `wgLayer/drawing` LPP:

- On the `NWell/drawing` LPP, with an enclosure of `0.8`.
- On the `Active/drawing` LPP, with an offset on `both` sides. The inner offset is specified as a fixed value and the outer offset as an expression.

techDeleteWaveguideDef

```
techDeleteWaveguideDef(
     d_waveguideDefId
     )
     => t / nil
```

Description

(Virtuoso Photonics Option) Deletes the specified waveguideDef object.

Arguments

d_waveguideDefId

The database ID of the waveguideDef object to be deleted.

Value Returned

`t`	The specified waveguideDef object was deleted.
`nil`	The specified waveguideDef object did not exist in the technology database.

Example

```
def=techFindWaveguideDefByLP(tf list("wgLayer" "drawing"))
techDeleteWaveguideDef(def)
```

Locates the waveguideDef object that has the specified LPP and deletes it.

techFindWaveguideDefByLP

```
techFindWaveguideDefByLP(
     d_techFileId
     tx_layer | (tx_layer tx_purpose)
     )
     => d_waveguideDefId / nil
```

Description

(Virtuoso Photonics Option) Returns the database ID of the waveguideDef object defined for the specified layer-purpose pair (LPP) in the specified technology database.

Arguments

d_techFileId	The database ID of the technology database to be searched.
tx_layer | (*tx_layer tx_purpose*)	The layer or LPP of the waveguideDef object. If the purpose is not specified or is `any`, the function searches for the waveguideDef object with the given layer and the `all` purpose. If the search fails, it searches for a waveguideDef object with the given layer and the `drawing` purpose.

Value Returned

d_waveguideDefId	The database ID of the `waveguideDef` object.
`nil`	A `waveguideDef` object could not be found, possibly because it has not been defined for the specified LPP in the technology database.

Example

```
techFindWaveguideDefByLP(tf list("wgLayer" "drawing"))
```

Finds the waveguideDef object in the `tf` technology database for the `wgLayer/drawing` LPP.

techHasWaveguideDefMinBendRadius

```
techHasWaveguideDefMinBendRadius(
     d_waveguideDefId
     )
     => t / nil
```

Description

(Virtuoso Photonics Option) Checks if a `minBendRadius` value is defined for a waveguideDef object.

Arguments

`d_waveguideDefId`

The database ID of the waveguideDef object to be checked.

Value Returned

`t`	The specified waveguideDef object has a `minBendRadius` value.
`nil`	The specified waveguideDef object could not be found.

Example

```
techHasWaveguideDefMinBendRadius(wg1)->t
```

Indicates that a `minBendRadius` value is defined for the waveguideDef object `wg1`.

techSetWaveguideDefMinBendRadius

```
techSetWaveguideDefMinBendRadius(
     d_waveguideDefId
     n_minBendRadius
     )
     => t / nil
```

Description

(Virtuoso Photonics Option) Sets the `minBendRadius` value for a waveguideDef object.

Arguments

d_waveguideDefId	The database ID of the waveguideDef object.
n_minBendRadius	The minimum bend radius of the object.

Value Returned

`t`	The specified `minBendRadius` value was set on the specified waveguideDef object.
`nil`	The specified waveguideDef object could not be found.

Example

```
techSetWaveguideDefMinBendRadius(wg1 0.3)->t
```

Sets `0.3` as the `minBendRadius` value for the waveguideDef object `wg1`.

10

LSW Layers Functions

The "LSW layers" SKILL functions operate on the `leRules` section of the technology database. The `leLswLayers` subsection of the `leRules` section lets you specify the layers that are initially displayed in the Palette assistant. In the `leLswLayers` subsection, you list the layers in the order in which you want them to appear in the Palette assistant.

If you do not define the `leLswLayers` subsection in the technology database, the `techLayerPurposePriorities` subsection of the `layerDefinitions` section determines the layers that are initially displayed and the order in which they are displayed in the Palette assistant.

techOrderLeLsw

```
techOrderLeLsw(
     tx_techfileLibName
     l_lswLayers
     )
     => t / nil
```

Description

Defines the relative priority for a dynamic set of layer-purpose pairs (LPPs) to be shown in the Palette assistant. The layers are pulled from the current `leLswLayers` section to the top of the display order according to their priority numbers. The smaller the number, the higher the priority and the higher the LPP is in the list.

This function is most useful when multiple technology databases are arranged in a technology graph. The LPPs specified are expected to exist in the graph rooted at the specified technology database. The function can be run on any variant of the process stack. It silently ignores any layers that are not present. Technology files on disk are not changed.

For more information about `leLswLayers`, see leLswLayers in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

tx_techfileLibName

The technology file library name.

l_lswLayers

An ordered list of LPPs to display in the Palette assistant. The list has the following syntax:

`list ( list(` *tx_layer tx_purpose n_priority*`)` `… )`

where,

- *tx_layer* is the layer name or number.
- *tx_purpose* is the purpose name or number.
- *n_priority* is the priority in which the specified LPPs appear in the display order.

Value Returned

`t`	The `leLSWLayers` subsection in the specified technology database was updated as specified.
`nil`	The technology database does not exist.

Example

Assume that the `leLswLayers` section has this sequence:

```
(("Via1"        "drawing")
 ("Via2"        "drawing")
 ("Via3"        "drawing")
 ("Via4"        "drawing")
 ("Via5"        "drawing")
 ("Metal1"      "drawing")
 ("Metal2"      "drawing")
 ("Metal3"      "drawing")
 ("Metal4"      "drawing")
 ("Metal5"      "drawing")
)
```

The function is called as follows:

```
techOrderLeLsw("topTech"
    list(
        list("Metal1" "drawing" 1)
        list("Metal2" "drawing" 2)
        list("Metal3" "drawing" 3)
        )
)
```

The `leLswLayers` section is updated as follows:

```
(("Metal1"       "drawing")
 ("Metal2"       "drawing")
 ("Metal3"       "drawing")
 ("Via1"         "drawing")
 ("Via2"         "drawing")
 ("Via3"         "drawing")
 ("Via4"         "drawing")
 ("Via5"         "drawing")
 ("Metal4"       "drawing")
 ("Metal5"       "drawing")
)
```

techSetLeLswLayers

```
techSetLeLswLayers(
     d_techID
     l_lswLayers
     )
     => t / nil
```

Description

Defines the layer display in the `leLswLayers` subsection of the `leRules` section in the specified technology database. If an `leLswLayers` subsection already exists, its contents are replaced with the specified data. If the layer display in the Palette assistant is currently controlled by `leLswLayers`, the list of layers is accordingly updated.

For more information about `leLswLayers`, see leLswLayers in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
l_lswLayers	An ordered list of layer-purpose pairs to display in the Palette assistant. The list has the following syntax: `list ( list( tx_layer tx_purpose ) … )` where, ■ *tx_layer* is the layer name or number ■ *tx_purpose* is the purpose name or number

Value Returned

`t`	The `leLSWLayers` subsection in the specified technology database was created or replaced.
`nil`	The technology database does not exist or one or more of the specified layer-purpose pairs are invalid.

Example

```
techSetLeLswLayers(techID
    list("metal1" "metal2" "metal3" "poly1" "pWell" "implant" "diff" "align")
    )
```

Creates an `leLswLayers` subsection with the specified layers and purposes in the technology database identified by `techID`.

techSetLeLswLayer

```
techSetLeLswLayer(
     d_techID
     l_layer
     )
     => t / nil
```

Description

Appends the specified layer-purpose pair at the end of the layer display definition in the `leLswLayers` subsection of the `leRules` section in the specified technology database. If the `leRules` section and the `leLswLayers` subsection do not exist, they are created with the specified data. If the layer display in the Palette assistant is currently controlled by `leLswLayers`, the list of layers is updated with the new layer-purpose pair.

For more information about the `leLswLayers` subsection of the technology file, see leLswLayers in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
l_layer	The layer-purpose pair to append to the `leLswLayers` subsection. Valid values: Layer name or number and purpose name or number

Value Returned

`t`	The layer-purpose pair was added to the `leLswLayers` subsection in the specified technology database.
`nil`	The technology database does not exist or the specified layer-purpose pair is invalid.

Example

```
techSetLeLswLayer(tfID list("nwell" "drawing"))
```

Appends the `nwell drawing` layer-purpose pair to the `leLswLayers` subsection in the `leRules` section of the technology database identified by `tfID`.

techGetLeLswLayers

```
techGetLeLswLayers(
     d_techID
     )
     => l_lswLayers / nil
```

Description

Returns the `leLswLayers` layers from the technology database identified by *`techID`*, if found. If `leLswLayers` is not found in the technology database identified by *`techID`*, the function searches any open technology databases in the graph for `leLswLayers` and returns the first set that is found. The function does not open any technology databases that are explicitly closed.

For more information about the `leLswLayers` subsection of the technology file, see leLswLayers in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

`d_techID`	The database identifier of the technology database.

Value Returned

`l_lswLayers`	A list of `leLswLayers` layers.
`nil`	The technology database or the `leLswLayers` subsection does not exist.

Example

```
techGetLeLswLayers( tfID )
=> (("metal1" "drawing")
    ("metal2" "drawing")
    ("metal3" "drawing")
    ("poly1" "drawing")
    ("pWell" "drawing")
    ("implant" "drawing")
    ("diff" "drawing")
    ("align" "drawing")
   )
```

Returns a list of layers defined in the `leLswLayers` subsection of the `leRules` section of the technology database identified by `tfID`.

techIsLeLswLayer

```
techIsLeLswLayer(
     d_techID
     l_layer
     )
     => t / nil
```

Description

Indicates whether the specified layer-purpose pair is listed in the layer display definition in the `leLswLayers` subsection of the `leRules` section in the specified technology database.

For more information about the `leLswLayers` subsection of the technology file, see leLswLayers in *Virtuoso Technology Data ASCII Files Reference*.

Arguments

d_techID	The database identifier of the technology database.
l_layer	The layer-purpose pair to check. Valid values: Layer name or number and purpose name or number

Value Returned

`t`	The specified layer-purpose pair is listed in the `leLswLayers` subsection in the specified technology database.
`nil`	The technology database does not exist or the layer-purpose pair is not listed in the `leLswLayers` subsection.

Example

```
techIsLeLswLayer(tfID list("metal1" "drawing"))
=> t
```

Finds the `metal1` `drawing` layer-purpose pair listed in the `leLswLayers` subsection of the `leRules` section in the technology database identified by `tfID`.

11

Display Resource File Functions

The display resource file functions are used to retrieve and modify the display resource data in virtual memory. To save the changes you make with these 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

Value Returned

t	The display device was deleted.
nil	The specified display device does not exist.

Example

```
drDeleteDisplay("psb")
=> t
```

Deletes the display device named 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. 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.

Value Returned

`t`	The specified color was deleted.
`nil`	The color does not exist for the specified display device.

Examples

```
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. 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.

Value Returned

t	The line style was deleted.
nil	The line style does not exist for the specified display device.

Examples

```
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. 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.

Value Returned

`t`	The specified packet was deleted.
`nil`	The packet does not exist for the specified display device.

Examples

```
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 device with 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. 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.

Value Returned

`t`	The stipple was deleted.
`nil`	The stipple does not exist for the specified display device.

Examples

```
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 device with 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 to which you want to save the display resource data.
g_saveChange	If set to `t`, saves only the changes made in virtual memory. If set to `nil`, saves all the display resource data from virtual memory. Valid values: `t`, `nil` Default: `nil`

Value Returned

`t`	The dump was successful.
`nil`	The dump was not successful.

Examples

```
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.

Value Returned

l_packetList	A list containing the display device name, packet name, and the 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.

Examples

```
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`.

drGetActiveDisplayDrf

```
drGetActiveDisplayDrf(
     )
     => t_activeDrf / nil
```

Description

Displays the absolute path of the currently active display resource file `display.drf`.

Arguments

None

Value Returned

t_activeDrf	The absolute path of the currently active `display.drf` file.
`nil`	No path returned.

Example

```
drGetActiveDisplayDrf()
"/servers/cic_vls/system/display.drf"
```

Returns the absolute path of the currently active `display.drf`.

drGetColor

```
drGetColor(
     tx_display
     tx_color
     )
     => l_colorList / nil
```

Description

Reads virtual memory and returns the display device name, color name, and the red, green, blue, and blink values for the color.

Arguments

tx_display	The display device name or identifier.
tx_color	The color name or index number.

Value Returned

l_colorList	A list containing the display device name, color name, and the red, green, blue, and blink values for the color.
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 for the specified display device name.

Arguments

t_displayName	The display device name.

Value Returned

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

Value Returned

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.

Value Returned

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 identifier `27832`.

drGetDisplayNameList

```
drGetDisplayNameList(
     )
     => l_displayNameList / nil
```

Description

Reads virtual memory and returns a complete list of display device names.

Arguments

None

Value Returned

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.

Value Returned

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.

Value Returned

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.

Value Returned

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*)

where,

- *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.

Value Returned

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.

Value Returned

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.

Value Returned

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.

Value Returned

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.

Value Returned

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 named `display.drf`) from any location.

Arguments

t_filename	The path and name of the display resource file (usually named `display.drf`).
g_askToSave	If set to `t`, prompts you to save your changes. Valid values: `t`, `nil` Default: `t`

Value Returned

`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_fillStyle ]
     )
     => 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.
t_fillStyle	The name of the fill style.

Value Returned

t	The packet was updated.
nil	The packet does not exist for the specified display device.

Examples

```
drSetPacket("psb" "bluethin_L" "blank" "thin" "blue" "tan" "stipple")
=> 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" "stipple")
=> 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.

12

MPT Functions

The multi-patterning technology (MPT) technology file SKILL functions are used to retrieve and modify the multi-patterning technology.

techGetIntegrationColorModel

```
techGetIntegrationColorModel(
     d_techFileId
     )
     => t_type / nil
```

Description

(Virtuoso Advanced Node for Layout Standard) Returns the value of the integration color model constraint for the current technology file. Possible return values are `any` or `locked`.

Arguments

d_techFileId	The database identifier of the technology file.

Value Returned

t_type	A string representing the integration color model (`any` or `locked`).
`nil`	The command was unsuccessful.

Example

```
techGetIntegrationColorModel( techFileId )
=> "locked"
```

Returns `locked` for the integration color model constraint.

techGetLayerNumColorMasks

```
techGetLayerNumColorMasks(
     d_techfile_Id
     xt_layer
     )
     => x_numColorMasks / nil
```

Description

(Virtuoso Advanced Node for Layout Standard) Returns the number of allowed color masks on the layer, if coloring is supported on it.

Arguments

d_techfile_Id	The database identifier of the technology file.
xt_layer	The layer name or number.

Value Returned

x_numColorMasks	Returns the number of allowed color masks on the layer. Valid values: `nil`, `2`, or `3`
`nil`	Indicates that coloring is not supported on the layer or the command was unsuccessful.

Examples

```
techGetLayerNumColorMasks(techFileId 10)
=> 2

techGetLayerNumColorMasks(techFileId "M1")
=> nil
```

techGetStdViaDefCutColoring

```
techGetStdViaDefCutColoring(
     d_viaDefID
     )
     => t_cutColoring / nil
```

Description

(Virtuoso Advanced Node for Layout Standard) Returns the cut coloring pattern to use for the cut patterns created for this type of `stdVia`.

Arguments

d_viaDefID	The database identifier of the via definition.

Value Returned

t_cutColoring	The cut coloring pattern on the via definition.
`nil`	The command was unsuccessful.

Example

```
cutColoring = techGetStdViaDefCutColoring( viaDefId )
```

techGetTechCutColoring

```
techGetTechCutColoring(
     d_techID
     )
     => t_defaultCutColoring / nil
```

Description

(Virtuoso Advanced Node for Layout Standard)(Virtuoso Advanced Node for Layout Standard) Returns the default cut coloring pattern to use for the cut patterns created for all of the `StdVias` defined in the technology file. The default checkerboard pattern is returned if the `cutPattern` coloring is not explicitly specified.

Arguments

d_techID	The database identifier of the technology database.

Value Returned

t_defaultCutColoring	The default cut coloring model for `stdViaDef` cuts.
`nil`	The command was unsuccessful.

Example

```
cutColoring = techGetTechCutColoring( tfId )
```

techIsStdViaDefCutColoringSet

```
techIsStdViaDefCutColoringSet(
     d_viaDefId
     )
     => t / nil
```

Description

(Virtuoso Advanced Node for Layout Standard) Returns a boolean value indicating whether the cut coloring pattern was explicitly set on the specified `stdViaDef`.

Arguments

d_viaDefId	The database identifier of the via definition of interest.

Value Returned

`t`	The cut coloring pattern was explicitly set on the `stdViaDef` of interest.
`nil`	The cut coloring pattern was not set on the `stdViaDef` of interest.

Example

```
isSet = techIsStdViaDefCutColoringSet( viaDefId )
```

techSetIntegrationColorModel

```
techSetIntegrationColorModel(
     d_techFileId
     t_type
     )
     => t / nil
```

Description

(Virtuoso Advanced Node for Layout Standard) Sets the value of the integration color model constraint for the current technology file. Valid values for the integration color model are `any` or `locked`.

Arguments

d_techFileId	The database identifier of the technology file.
t_type	A string representing integration color model (`any` or `locked`).

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Examples

```
techSetIntegrationColorModel(techFileId "any")
=> t
techSetIntegrationColorModel(techFileId "unlocked")
=> nil   ; no "unlocked" color model
```

techSetLayerNumColorMasks

```
techSetLayerNumColorMasks(
     d_techfile_Id
     xt_layer
     x_numColorMasks
     )
     => t / nil
```

Description

(Virtuoso Advanced Node for Layout Standard) Assigns the number of allowed color masks on the specified layer. `0`, `2`, and `3` are valid numbers that can be set. This function does not allow you to set the attribute on a DFII system-reserved layer.

Arguments

d_techfile_Id	The database identifier of the technology file.
xt_layer	The layer name or number.
x_numColorMasks	The number of color masks to be set. Valid values: `0`, `2`, or `3` The value `0` indicates coloring is not supported on the layer.

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Examples

```
techSetLayerNumColorMasks(techFileId "M1" 1)
=> nil

techSetLayerNumColorMasks(techFileId "M1" 2)
=> t

techSetLayerNumColorMasks(techFileId "Unrouted" 2)
=> nil
```

techSetStdViaDefCutColoring

```
techSetStdViaDefCutColoring(
     d_viaDefID
     t_cutColoring
     )
     => t / nil
```

Description

(Virtuoso Advanced Node for Layout Standard) Sets the cut coloring pattern to use for the cut patterns created for the specified type of `stdVia`.

Arguments

d_viaDefID	The `stdViaDef` of interest.
d_cutColoring	The cut coloring pattern to be set for the `stdViaDef`.

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Example

```
techSetStdViaDefCutColoring(viaDefId "alternatingRows")
```

techSetTechCutColoring

```
techSetTechCutColoring(
     d_techID
     t_defaultCutColoring
     )
     => t / nil
```

Description

(Virtuoso Advanced Node for Layout Standard) Sets the default cut coloring pattern to use for the cut patterns created for all `stdVias` defined in the technology file.

Arguments

d_techID	The database identifier of the technology database.
t_defaultCutColoring	The cut coloring pattern to be set. Valid values: `checkerboard` and `alternatingRows`

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

Example

```
techSetTechCutColoring(tdId "alternatingRows")
```

13

Display Form Functions

The display form SKILL functions that can be used to display technology file forms.

techManagerOpenTechToolBox

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

Description

Opens the Technology Tool Box form. It is equivalent to the CIW *Tools – Technology File Manager* command.

Arguments

None

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

techManagerOpenDisplayToolBox

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

Description

Opens the Display Resources Tool Box. It is equivalent to the CIW *Tools – Display Resource Manager* command.

Arguments

None

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

tcDisplayNewTechForm

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

Description

Opens the New Technology Library form. It is equivalent to clicking the *New* command in the Technology Tool Box form.

Arguments

None

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

tcNewLibDisplayRefTechForm

```
tcNewLibDisplayRefTechForm(
     t_newLibName
     )
     => t / nil
```

Description

Opens the Reference Existing Technology Libraries form that displays the reference technology file library choices for the newly created technology library with the specified name.

You can also open this form by clicking the *Set Reference* button in the Technology Tool Box form.

Arguments

t_newLibName	The name for the new technology library.

Value Returned

t	The command was successful.
nil	The command was unsuccessful.

Example

```
tcNewLibDisplayRefTechForm("MyTechLibrary")
```

tcDisplayTechGraphForm

```
tcDisplayTechGraphForm(
     [ d_techID ]
     )
     => t / nil
```

Description

Opens the Technology Database Graph form. It is equivalent to clicking the *Graph* button in the Technology Tool Box form.

Arguments

d_techID	The technology database identifier.

None

Value Returned

t	The command was successful.
nil	The command was unsuccessful.

Example

```
cvId = geGetWindowCellView()
tfId = techGetTechFile( cvId )
tcDisplayTechGraphForm( tfId )
```

Gets the ID of the open cellview and stores it in `cvID`. Next, it retrieves the technology file ID by using `cvId`, and then uses that ID to open the Technology Database Graph form.

tcDisplayAttachTechForm

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

Description

Opens the Attach Technology Library to Design Library form. It is equivalent to clicking the *Attach* button in the Technology Tool Box form.

Arguments

None

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

tcDisplayLoadTechForm

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

Description

Opens the Load Technology File form. It is equivalent to clicking the *Load* button in the Technology Tool Box form. It is also equivalent to the `tcDisplayCompTechForm` SKILL function.

Arguments

None

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

tcDisplayCompTechForm

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

Description

Opens the Load Technology File form. It is equivalent to clicking the *Load* button in the Technology Tool Box form. It is also equivalent to the `tcDisplayLoadTechForm` SKILL function.

Arguments

None

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

tcDisplayDumpTechForm

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

Description

Opens the Dump Technology File form. It is equivalent to clicking the *Dump* button in the Technology Tool Box form.

Arguments

None

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

tcDisplayDiscardTechForm

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

Description

Opens the Discard Edits To Technology File form. It is equivalent to clicking the *Discard* button in the Technology Tool Box form.

Arguments

None

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

tcDisplaySaveTechForm

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

Description

Opens the Save Technology File form. It is equivalent to clicking the *Save* button in the Technology Tool Box form.

Arguments

None

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

tcQcInstallDevices

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

Description

Opens the Install Device form. This is equivalent to clicking the *Install Device* button in the Technology Tool Box form.

Arguments

None

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

dreInvokeDre

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

Description

Opens the Display Resource Editor form. It is equivalent to clicking the *Edit* button in the Display Resources Tool Box.

Arguments

None

Value Returned

`t`	The command was successful.
`nil`	The command was unsuccessful.

14

SnapPatternDefinition Functions

The snap pattern definition functions are used to create and find snap pattern definitions in the technology database. They are categorized as:

- Snap Pattern Definition SKILL Functions
- Width Spacing Pattern SKILL Functions
- Width Spacing Pattern Groups SKILL Functions
- Width Spacing Snap Pattern Def SKILL Functions
- Related Snap Patterns SKILL Functions

Snap Pattern Definition SKILL Functions

The snap pattern definition functions are used to create and find snap pattern definitions in the technology database.

techCreateSnapPatternDef

```
techCreateSnapPatternDef(
     d_techFileId
     tx_name
     (tx_layer tx_purpose)
     t_stepDirection
     n_step
     [ l_snappingLayers ]
     [ t_type ]
     [ n_offset ]
     [ n_trackWidth ]
     [ l_trackGroups ]
     )
     => d_snapPatternDefId / nil
```

Description

(Virtuoso Advanced Node for Layout Standard) Creates a snapPatternDef by using the specified parameters.

Arguments

d_techFileId	The ID of the technology file in which the snapPatternDef is to be created.
tx_name	The name of the snapPatternDef.
tx_layer	The layer name or number of the snapPatternDef.
tx_purpose	The purpose name or number of the snapPatternDef. The parent of this purpose must be "annotation".
t_stepDirection	The direction in which *n_step* is applied. Valid values: `vertical`, `horizontal`
n_step	The distance between the pattern tracks of the snapPatternDef.

l_snappingLayers
A list of the layers, purposes, and enclosures that determine the shapes that snap to the snap pattern. Each entry is of the form:

```
(tx_snapLayer l_trackEnclosure [l_purposes]
[l_exceptOverlapLPPs]
["multiTrackCenter"|"singleTrackCenter"]
```

If *l_purposes* is not specified, then all shapes on *tx_snapLayer* snap to this snapPatternDef. The snapping is done by aligning the shapes on *tx_snapLayer* to the grid using one of the enclosures specified in *l_trackEnclosure*.

tx_snapLayer
A list of layers. Shapes on *l_snappingLayers* of a specific layer snap to a snap pattern. Applies to all layers if *tx_snapLayer* is not specified.

The list is specified in this format:

```
l_snappingLayers ( tx_snapLayer … )
```

l_trackEnclosure
A list of enclosures of a snapping shape beyond a grid line in the snap pattern direction. If vias and top-level shapes are present on the snapping layer (and snapping purpose, if used), they are snapped to the snap pattern grid by using this value.

The list is specified in this format:

```
l_trackEnclosures ( g_range … )
```

l_purposes
A list of purposes. Shapes on *l_snappingLayers* with a specific purpose snap to a snap pattern. Applies to all purposes if `l_purposes` is not specified.

The list is specified in this format:

```
l_purposes ( tx_snapPurpose … )
```

l_exceptOverlapLPPs
A list of layer-purpose name pairs. Applicable only when *t_type* is `global`. Snapping does not happen if a shape is completely inside the shape defined by *l_exceptOverlapLPPs*. If the shapes overlap partially, snapping does take place.

The list is specified in this format:

```
((t_layerName t_purposeName)…)
```

`"multiTrackCenter"|"singleTrackCenter"`

The snapping mode indicated by one of these values:

- `multiTrackCenter`: Default value used when the attribute is not specified. It indicates if a shape can be centered on any number of tracks in a way that when the *`l_trackEnclosure`* value is met on both sides, it is snapped to the grid. When a shape is created, it is assumed that all edges will be on the grid and the first and second points always snap to the grid.

  If the combined snapping layer BBox inside an instance can be centered in a way that the *`l_trackEnclosure`* value is met on both sides, then the instance snaps to the grid using that snapping layer. If no snapping layer BBox inside an instance snaps to the grid, then the instance is not snapped.

- `singleTrackCenter`: Indicates if a shape can be centered on a single track in a way that when the `trackEnclosure` value is met on both sides, it is snapped to the grid. When a shape is created, it is not assumed that all edges will be on the grid. Only a shape with a dimension that is twice the *`l_trackEnclosure`* value would snap to the grid.

  In this case, the combined snapping layer BBox is not considered. If the device contains any shape meeting the dimension rule, the shape is snapped even if the combined layer BBox does not meet the enclosure on both sides.

`t_type`

The type of this snapPatternDef.

Valid values: `local` (default), `global`

A `global` snapPatternDef can be applied to a cellview through a constraint.

A `local` snapPatternDef is only active if a shape on its LPP is drawn in the layout.

`n_offset`

The distance from the nearest period track to the anchor reference (either the lower edge of the PRBoundary or the origin axis).

n_trackWidth	The width of a snap pattern track. This is only for visualization.
l_trackGroups	(Virtuoso Advanced Node for Layout Only) A list of track groups. Each entry is of the form: (*n_numTracks x_space*) where, ■ *n_numTracks* is the number of tracks in each track group. ■ *x_space* is the spacing to the next track group.

Value Returned

d_snapPatternDefId	The ID of the created snapPatternDef.
nil	Returned in case of failure.

Example

```
techCreateSnapPatternDef(
    tech
    "gridDef"
    list("Gridlayer" "type1")
    "vertical"
    0.4
    '(("Metal1" (0.045) ("drawing") (("Metal7" "drawing")) "singleTrackCenter"))
    "global"
    0.1
    0.2
    )
```

Creates a snapPatternDef for LPP `Gridlayer/type1`. The grid has a vertical direction with grid space `0.4`, offset `0.1`, and trackWidth `0.2`. Shapes on `Metal1/drawing` snap to the snap pattern. No snapping takes place for `Metal1/drawing` shapes that fully overlap a `Metal7/drawing` shape. The track enclosure for `Metal1` is `0.045`. The snapping mode is `singleTrackCenter`.

techDeleteSnapPatternDef

```
techDeleteSnapPatternDef(
     d_techSnapPatternDefId
     )
     => t / nil
```

Description

(Virtuoso Advanced Node for Layout Standard) Deletes the specified `snapPatternDef` object.

Arguments

`d_techSnapPatternDefId`

Specifies the ID of the `snapPatternDef` object to be deleted.

Value Returned

`t`	Specified `snapPatternDef` object was deleted.
`nil`	Returned in case of failure.

Example

```
techDeleteSnapPatternDef( spdef )
```

Deletes the `snapPatternDef` object with the ID `spdef`.

techFindSnapPatternDefByLP

```
techFindSnapPatternDefByLP(
     d_techID
     (tx_layer tx_purpose)
     )
     => snapPatternDefId / nil
```

Description

(Virtuoso Advanced Node for Layout Standard) Finds a snapPatternDef in the current technology database by layer or purpose.

Arguments

d_techID	The database identifier of the technology database.
tx_layer	The layer name or number.
tx_purpose	The purpose name or number.

Value Returned

snapPatternDefId	The snapPatternDef is returned in the specified technology database.
nil	Unable to find snapPatternDef.

Example

```
techFindSnapPatternDefByLP(tech list("FF" "polygrid"))
```

Returns the snapPatternDef based on database identifier, `tech list` on layer, `FF` and purpose, `polygrid`.

techFindSnapPatternDefByName

```
techFindSnapPatternDefByName(
     d_techID
     t_name
     )
     => snapPatternDefId / nil
```

Description

(Virtuoso Advanced Node for Layout Standard) Finds a snapPatternDef in the current technology database by name.

Arguments

d_techID	The database identifier of the technology database.
t_name	The name of snapPatternDef.

Value Returned

snapPatternDefId	The `snapPatternDef` is returned in the specified technology database.
`nil`	Unable to find `snapPatternDef`.

Example

```
techFindSnapPatternDefByName(tech "test1")
```

Returns the snapPatternDef based on database identifier, `tech`, and name, `test1`.

Width Spacing Pattern SKILL Functions

The width spacing pattern functions are used to create, find, and access widthSpacingPattern objects the technology database.

techCreateWidthSpacingPattern

```
techCreateWidthSpacingPattern(
     d_techFileId
     t_name
     l_patternSpecs
     [ g_offset ]
     [ b_repeatOffset ]
     [ b_shiftColor ]
     [ t_allowedRepeatMode ]
     [ t_defaultRepeatMode ]
     )
     => d_WidthSpacingPatternId / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Creates a width spacing pattern in the specified technology file.

Arguments

d_techFileId	Specifies the ID of the technology file in which the width spacing pattern is to be created.
t_name	Specifies the width spacing pattern name.

l_patternSpecs

Specifies a list of track group specifications.

`list (`*l_patternSpec*`)`

where,

l_patternSpec is a track group specification.

l_patternSpec`: list(`*l_specs* `[`*d_repeat* `[`*l_wireTypes* `[`*l_colorNames*`] [`*l_displayPacketNames*`]])`

- *l_specs* is the list of individual track specifications for the specified track group.

  `list(`*l_spec*`)`

  l_spec is the track specification.

  l_spec`: list(`*g_width g_space* `[`*t_wireType* `[`*t_colorName* `[`*t_displayPacketName*`]])`

  - *g_width* is the width of shapes on this track.
  - *g_space* is the center-line distance of the next track from the current track, in the period direction.
  - *t_wireType* is a string that represents the wire type. It can only be specified if *d_repeat l_wireTypes* is not specified.

    Default value is an empty string (`""`).
  - *t_colorName* is the color of the specified track. It can only be specified if *d_repeat l_wireTypes* is not specified. By default, tracks are not colored.

    Valid values: `mask1Color`, `mask2Color`, and `mask3Color`.
  - *t_displayPacketName* is the name of a display packet that can be specified to control how the track should be drawn on the screen. It can be specified only if *t_colorName* is specified.

- *d_repeat* is a number of repeats for the track group. A value of 1 creates the track group from the specified *l_specs* (1x). A value of 2 creates the track group from the specified *l_specs* that are repeated once (2x).

  Default value is 1.

  If specified, *t_wireType* and *t_colorName* cannot be specified in the *l_specs*.

- *l_wireTypes* is a list of wire types for the track group (only if *d_repeat* is specified).

  l_wireTypes: list([*t_wireType* …])

  Note: An empty string in the list can be used if you do not want to specify a wire type for a track. If the number of tracks in the track group is greater than the number of wire types in this list, then the wire types list is repeated across the track group until all tracks are assigned a wire type, one-by-one.

  Default value: An empty list.

  - *t_wireType* is a string that represents the wire type.

  For example, for a track group with 6 tracks:

  - If *l_wireTypes* is `list("a" "b")`, then tracks 1, 3, and 5 are wire type **a**; tracks 2, 4, and 6 are wire type **b**.
  - If *l_wireTypes* is `list("a" "a" "b")`, then tracks 1, 2, 4, and 5 are wire type **a**; tracks 3 and 6 are wire type **b**.

- *l_colorNames* is a list of track colors for the track group (only if *d_repeat* is specified).

  Default value: An empty list.

  l_colorNames: list([*t_colorName* …])

Note: The `grayColor` entry can be used if you do not want to specify a color for a track. If the number of tracks in the specified track group is greater than the number of colors in this list, then the color list is repeated across the track group until all tracks are assigned a color, one-by-one.

t_colorName is a color of the specified track.

Valid values: `mask1Color`, `mask2Color`, `mask3Color`, and `grayColor`.

- *l_displayPacketNames* is a list of display packet names for the track group (only if *d_repeat* is specified).

  Default value: An empty list.

  l_displayPacketName:
  `list([`*t_displaypacketName* `…])`

  t_displayPacketName is the display packet name for the specified track.

g_offset

Specifies the distance of the first track from the period track.

Default value: 0

b_repeatOffset

Specifies a flag to indicate whether the offset is applied when a pattern is repeated.

Default value: `nil`

b_shiftColor

Specifies a flag to indicate whether track colors are shifted when a pattern is repeated. This attribute is used only when all tracks are individually colored. No automatic color assignment is done. If no color is specified for a track, it will be gray.

Default value: `nil`

t_allowedRepeatMode

Specifies the allowed repeat mode.

Valid values: `any` (default), `none`, `steppedOnly`, and `flippedOnly`.

t_defaultRepeatMode	Specifies the default repeat mode. Valid values: `noRepeat` (default), `stepped`, `flippedStartsWithOdd`, and `flippedStartsWithEven`.

Value Returned

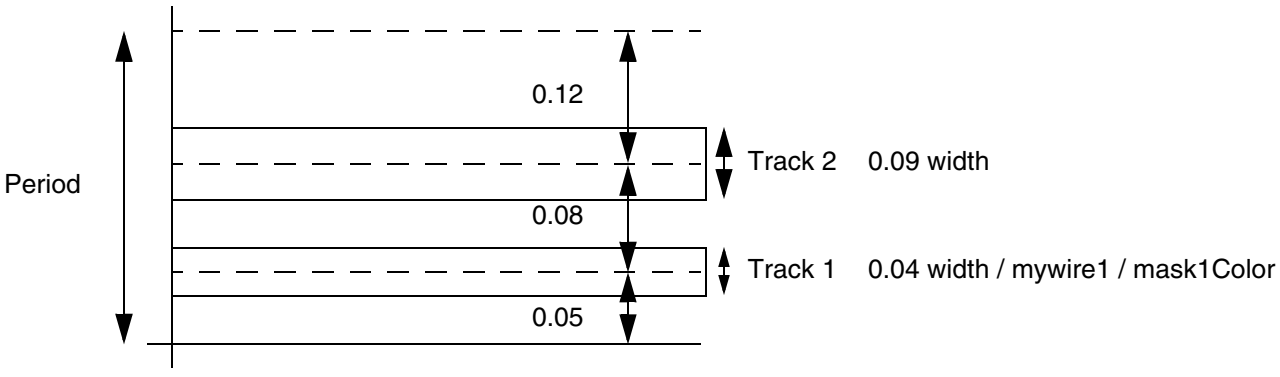
d_widthSpacingPatternId	Returns the ID of the created width spacing pattern.
`nil`	Returned in case of failure.

Examples

Example 1

```
techCreateWidthSpacingPattern(
    tech
    "wsp"
    '((((0.04 0.08 "mywire1" "mask1Color") (0.09 0.12))))
    0.05
    )
```

Creates `wsp` width spacing pattern in the specified technology file as shown in the figure below.



Example 2

```
techCreateWidthSpacingPattern(tech "wsp1"
    '(
        (
            (
                (0.04 0.08)
            )
            4
            ("vdd1" "sig1")
            ("mask2Color" "mask3Color")
        )
        (
            (
                (0.04 0.08 nil "mask2Color")
                (0.09 0.012 "big")
            )
        )
    )
    0.007
    t
    nil
    "steppedOnly"
    "stepped"
)
```

In this example, a width spacing pattern `wsp1` is created in the technology database tech with two track groups, defining a total of six tracks.

- For the first track group:
  - The width of the track is `0.04`.
  - The distance of the next track from the current track, in the period direction is `0.08`.
  - The track group is repeated `4` times resulting in a total of 4 tracks.
  - The offset distance of the first track from the period track is `4`.
  - The wire type of the first and third track is `vdd1`. The wire type of the second and fourth track is `sig1`.
  - The track color of the first and third track is `mask2Color`. The track color of the second and fourth track is `mask3Color`.
- For the second track group:
  - For the first track in this group:
    - The width is `0.04`.
    - The distance of the next track from this track, in the period direction is `0.08`.
    - No wire type is specified.

- The track color is `mask2Color`.

- For the second track in this group and the sixth track in the pattern:
  - The width is `0.09`.
  - The distance of the next track from this track, in the period direction is `0.012`.
  - The wire type is `big`.
  - No track color is specified.

- The second track group is not repeated, and the wire type and track color of each track are specified on individual tracks.

- The distance of the first track from the period grid is 0.007.
- The `repeatOffset` attribute is set to `t`. This means the height of the pattern includes the offset of the first track.
- The `shiftColor` attribute is set to `nil`. This means the track colors are not shifted when the pattern repeats.
- *`allowedRepeatMode`* is `steppedOnly`, which implies that the pattern can only be stepped, but not flipped. In conformance, *`defaultRepeatMode`* is `stepped`.

techCreateWidthSpacingPatternWithColor

```
techCreateWidthSpacingPatternWithColor(
     d_techFileId
     t_name
     l_patternSpecs
     t_startingColor
     [ g_offset ]
     [ b_repeatOffset ]
     [ t_allowedRepeatMode ]
     [ t_defaultRepeatMode ]
     )
     => d_WidthSpacingPatternId / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Creates a width spacing pattern in the specified technology file. This function requires you to specify the color for only the first track, referred to as the starting color. All other tracks are colored automatically by shifting colors.

Arguments

d_techFileId	Specifies the ID of the technology file in which the width spacing pattern is to be created.
t_name	Specifies the width spacing pattern name.

l_patternSpecs

Specifies a list of track group specifications.

list(*l_patternSpec*)

where,

l_patternSpec is a track group specification.

l_patternSpec: list(*l_trackSpecs* [*d_repeat* [*l_trackWireTypes*] [*l_displayPacketNames*]])

- *l_trackSpecs* is the list of individual track specifications for the specified track group.

  list(*l_trackSpec*)

  l_trackSpec is a track specification.

  l_trackSpec: list(*g_width g_space* [*t_wireType*][*t_displayPacketName*])

  - *g_width* is a width of the shapes for the specified track.
  - *g_space* is the distance of the next track from this track, in the period direction.
  - *t_wireType* is a string that represents the wire type (only if *d_repeat l_trackWireTypes* is not specified).
  - *t_displayPacketName* is the name of a display packet that can be specified to control how the track should be drawn on the screen.

- *d_repeat* is the number of repeats for the track group. A value of 1 creates the track group from the specified *l_specs* (1x). A value of 2 creates the track group from the specified *l_specs* that are repeated once (2x).

- *l_trackWireTypes* is a list of wire types for the track group (only if *d_repeat* is specified on the track group).

  list([*t_wireType* …])

  To not assign a wire type to an individual track, an empty string can be used.

  t_wireType is a string that represents the wire type.

- *l_displayPacketNames* is a list of display packet names for the track group (only if *d_repeat* is specified).

  Default value: An empty list.

  l_displayPacketName:
  list([*t_displaypacketName* …])

  t_displayPacketName is the display packet name for the specified track.

t_startingColor — Specifies the color of the first track. All tracks are automatically colored by color shifting from the previous track color.

Valid values: `mask1Color`, `mask2Color`, `mask3Color`, and `grayColor` (uncolored).

g_offset — Specifies the distance of the first track from the period track.

Default value: `0`

b_repeatOffset — Specifies a flag to indicate whether the offset is applied when a pattern is repeated.

Default value: `nil`

t_allowedRepeatMode — Specifies the allowed repeat mode.

Valid values: `any` (default), `none`, `steppedOnly`, and `flippedOnly`.

t_defaultRepeatMode — Specifies the default repeat mode.

Valid values: `noRepeat` (default), `stepped`, `flippedStartsWithOdd`, and `flippedStartsWithEven`.

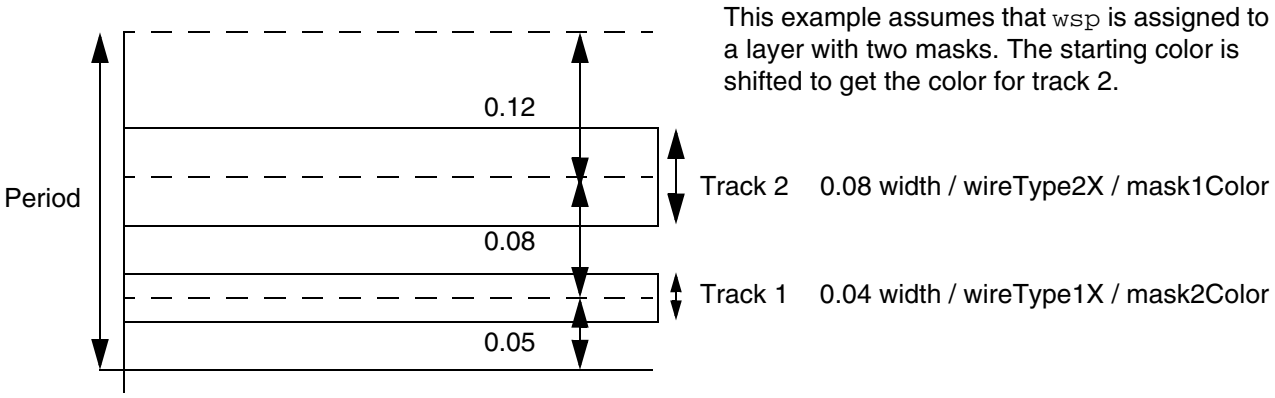
Value Returned

d_widthSpacingPatternId	Returns the ID of the created width spacing pattern.
nil	Returned in case of failure.

Example

```
techCreateWidthSpacingPatternWithColor(
    tech
    "wsp"
    list(list(list(list(0.04 0.08 "wireType1X") list(0.08 0.12 "wireType2X"))))
    "mask2Color" 0.05
    "steppedOnly" "stepped"
    )
```

Creates the wsp width spacing pattern with two tracks in the specified technology file, as shown in the following figure. *allowedRepeatMode* is steppedOnly, which implies that the pattern can only be stepped, but not flipped. In conformance, *defaultRepeatMode* is stepped.



techDeleteWidthSpacingPattern

```
techDeleteWidthSpacingPattern(
     d_techWSPId
     )
     => t / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Deletes the specified width spacing pattern.

Arguments

d_techWSPId	Specifies the ID of the width spacing pattern to be deleted.

Value Returned

`t`	Specified width spacing pattern was deleted.
`nil`	Returned in case of failure.

Example

```
techDeleteWidthSpacingPattern( wsp1 )
```

Deletes the width spacing pattern with ID `wsp1`.

techFindWidthSpacingPattern

```
techFindWidthSpacingPattern(
     d_techFileId
     t_name
     )
     => d_WidthSpacingPatternId / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Searches for the width spacing pattern with the specified name in the specified technology file or a referenced technology database in an ITDB graph of the specified technology database.

Arguments

d_techFileId	Specifies the ID of the technology file in which the search is to be performed.
t_name	Specifies the width spacing pattern name.

Value Returned

d_widthSpacingPatternId	Returns the ID of the width spacing pattern, if found.
nil	Returned in case of failure.

Example

```
techFindWidthSpacingPattern(tech "wsp1")
```

Returns the ID of the width spacing pattern with the name `wsp1`, if found.

techGetWidthSpacingPatternAllowedRepeatMode

```
techGetWidthSpacingPatternAllowedRepeatMode(
     d_wspId
     )
     => t_allowedRepeatMode / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Returns the allowed repeat mode set for a width spacing pattern.

Arguments

d_wspId	Specifies the ID of the width spacing pattern.

Value Returned

t_allowedRepeatMode	Returns the allowed repeat mode set for the width spacing pattern.
nil	Returned in case of failure.

Example

```
techGetWidthSpacingPatternAllowedRepeatMode( wspId )
```

Returns the allowed repeat mode for the width spacing pattern.

techGetWidthSpacingPatternDefaultRepeatMode

```
techGetWidthSpacingPatternDefaultRepeatMode(
     d_wspId
     )
     => t_defaultRepeatMode / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Returns the default repeat mode set for a width spacing pattern.

Arguments

d_wspId	Specifies the ID of the width spacing pattern.

Value Returned

t_defaultRepeatMode	Returns the default repeat mode set for the width spacing pattern.
nil	Returned in case of failure.

Example

```
techGetWidthSpacingPatternDefaultRepeatMode( wspId )
```

Returns the default repeat mode for the width spacing pattern.

techGetWidthSpacingPatterns

```
techGetWidthSpacingPatterns(
     d_techFileId
     )
     => d_widthSpacingPatternIds / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Returns a list of all the width spacing pattern IDs in the specified technology file or in ITDB graph of the specified technology database.

Arguments

d_techFileId	Specifies the ID of the technology file in which the listing is performed.

Value Returned

d_widthSpacingPatternId	Returns a list of width spacing pattern IDs.
nil	Returned in case of failure.

Example

```
techGetWidthSpacingPatterns( tech )
```

Returns a list of all width spacing pattern IDs found in the specified technology file or in ITDB graph of the specified technology database.

techSetWidthSpacingPatternRepeatMode

```
techSetWidthSpacingPatternRepeatMode(
     d_wspId
     t_allowedRepeatMode
     t_defaultRepeatMode
     )
     => t / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Sets the allowed and default repeat modes for a width spacing pattern.

Arguments

d_wspId	Specifies the ID of the width spacing pattern.
t_allowedRepeatMode	Specifies the allowed repeat pattern mode. Valid values: `any`, `none`, `steppedOnly`, and `flippedOnly`
t_defaultRepeatMode	Specifies the default repeat pattern mode. Valid values: `stepped`, `flippedStartsWithOdd`, and `flippedStartsWithEven`

Value Returned

`t`	The specified allowed and default repeat modes were set.
`nil`	Returned in case of failure.

Example

```
techSetWidthSpacingPatternRepeatMode(wspId "steppedOnly" "stepped")
```

Sets the allowed repeat mode to `steppedOnly` and the default repeat mode to `stepped`.

Width Spacing Pattern Groups SKILL Functions

The width spacing pattern groups SKILL functions are used to create, find, and access widthSpacingPatternGroup objects in the technology database.

techCreateWidthSpacingPatternGroup

```
techCreateWidthSpacingPatternGroup(
     d_techFileId
     t_name
     l_patternNames
     )
     => d_WidthSpacingPatternGroupId / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Creates a group of width spacing patterns in the specified technology file.

Arguments

d_techFileId	Specifies the ID of the technology file in which the width spacing pattern group is to be created.
t_name	Specifies the name for the width spacing pattern group.
l_patternNames	Specifies the list of width spacing pattern names to include in the group.

Value Returned

d_widthSpacingPatternGroupId	Returns the ID of the created width spacing pattern group.
nil	Returned in case of failure.

Example

```
techCreateWidthSpacingPatternGroup(
    tech
    "group1"
    '("wsp1" "wsp2")
    )
```

Returns the ID of the `group1` group of width spacing patterns found in the specified technology file.

techDeleteWidthSpacingPatternGroup

```
techDeleteWidthSpacingPatternGroup(
     d_techWSPGroupId
     )
     => t / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Deletes the specified width spacing pattern group.

Arguments

d_techWSPGroupId	Specifies the ID of the width spacing pattern group to be deleted.

Value Returned

`t`	Specified width spacing pattern group was deleted.
`nil`	Returned in case of failure.

Example

```
techDeleteWidthSpacingPatternGroup( wsp_group1 )
```

Deletes the width spacing pattern group with ID `wsp_group1`.

techFindWidthSpacingPatternGroup

```
techFindWidthSpacingPatternGroup(
     d_techFileId
     t_name
     )
     => d_widthSpacingPatternGroupId / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Searches for the width spacing pattern group with the specified name in the specified technology file or a referenced technology database in an ITDB graph of the specified technology database.

Arguments

d_techFileId	Specifies the ID of the technology file in which the search is to be performed.
t_name	Specifies the name of the width spacing pattern group.

Value Returned

l_widthSpacingPatternGroupIds	Returns the ID of the width spacing pattern group, if found.
nil	Returned in case of failure.

Example

```
techFindWidthSpacingPatternGroup(tech "wsp_group1")
```

Returns the ID of the `wsp_group1` width spacing pattern group found in the specified technology file.

techGetWidthSpacingPatternGroups

```
techGetWidthSpacingPatternGroups(
     d_techFileId
     )
     => l_widthSpacingPatternGroupIds / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Returns a list of all the width spacing pattern group IDs in the specified technology file.

Arguments

d_techFileId	Specifies the ID of the technology file in which the listing is performed.

Value Returned

l_widthSpacingPatternGroupIds	Returns a list of width spacing pattern group IDs.
nil	Returned in case of failure.

Example

```
techGetWidthSpacingPatternGroups( tech )
```

Returns a list of all width spacing pattern group IDs found in the specified technology file.

Width Spacing Snap Pattern Def SKILL Functions

The width spacing snap pattern definition functions are used to create, find, access, and delete widthSpacingSnapPatternDefs in the technology database.

techCreateWidthSpacingSnapPatternDef

```
techCreateWidthSpacingSnapPatternDef(
     d_techFileId
     t_name
     l_lp
     t_direction
     g_period
     t_defaultPatternName
     l_snappingLayers
     [ g_offset ]
     [ l_patternNames ]
     [ l_patternGroupNames ]
     [ g_gridType ]
     )
     => d_widthSpacingSnapPatternDefId / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Creates a `widthSpacingSnapPatternDef` object in the specified technology file.

Arguments

d_techFileId	Specifies the ID of the technology file in which the `widthSpacingSnapPatternDef` object is to be created.
t_name	Specifies the name for the `widthSpacingSnapPatternDef` object.
l_lp	Specifies the layer and purpose used to draw regions for this `widthSpacingSnapPatternDef` in a layout. *l_lp*: `list(`*tx_layer tx_purpose*`)`
t_direction	Specifies the pattern direction. Valid values: `horizontal` or `vertical`.
g_period	Specifies the spacing between coarse-grain period tracks.
t_defaultActivePattern	Specifies the pattern, which is used in areas where no region has been drawn.

l_snappingLayers

Specifies the list of snapping layer definitions.

list(*l_snappingLayer*)

where,

- *l_snappingLayer* is a snapping layer definition.

  l_snappingLayer: list(*tx_layer [l_purposeList]*)

- *tx_layer* is snap layer name.
- *l_purposeList* is a list of purpose names.

  l_purposeList: list(*tx_purpose*)

g_offset

Specifies the distance of the nearest period track to the anchor reference (either the lower edge of the PRBoundary or the origin axis).

Default value: 0

l_patternNames

Specifies a list of allowed width spacing pattern names.

list(*t_patternName*)

l_patternGroupNames

Specifies a list of allowed width spacing pattern group names.

list(*t_patternGroupName*)

g_gridType

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

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

Value Returned

d_widthSpacingSnapPatternDefId	Returns the ID of the created object.
nil	Returned in case of failure.

Example

```
techCreateWidthSpacingSnapPatternDef(
    tech
    "wsspDef"
    '("layer1"  "purpose1")
    "vertical" 0.2
    "wsp3"
    '(("layer2" ("dummy1" "dummy2")) ("layer3" ("dummy3" "dummy4")))
    0.1
    '("wsp" "wsp2")
    '("basePatterns" "wspg2") )
```

Creates the `wsspDef` object in the specified technology file.

techDeleteWidthSpacingSnapPatternDef

```
techDeleteWidthSpacingSnapPatternDef(
     d_techWidthSpacingSnapPatternDefId
     )
     => t / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Deletes the specified widthSpacingSnapPatternDef object.

Arguments

d_techWidthSpacingSnapPatternDefId

Specifies the ID of the `widthSpacingSnapPatternDef` object to be deleted.

Value Returned

`t`	Specified `widthSpacingSnapPatternDef` object was deleted.
`nil`	Returned in case of failure.

Example

```
techDeleteWidthSpacingSnapPatternDef( wsspDef )
```

Deletes the widthSpacingSnapPatternDef object with ID `wsspDef`.

techGetWidthSpacingSnapPatternDefsByLP

```
techGetWidthSpacingSnapPatternDefsByLP(
     d_techFileId
     l_LP
     )
     => l_widthSpacingSnapPatternDefIDList / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Retrieves from the specified technology file a list of widthSpacingSnapPatternDefs defined in a layer-purpose pair.

Arguments

d_techFileId	Specifies the ID of the technology file in which the search is to be performed.
l_LP	List of layer and purpose names.

Value Returned

l_widthSpacingSnapPatternDefIDList	A list of `widthSpacingSnapPatternDef` IDs defined in the given layer-purpose pair.
`nil`	Returned in case of failure.

Example

```
techGetWidthSpacingSnapPatternDefsByLP(tech list("layer1" "purpose1"))
```

Returns a list of widthSpacingSnapPatternDefs defined for `layer1` and `purpose1` in the specified technology file.

techFindWidthSpacingSnapPatternDefByName

```
techFindWidthSpacingSnapPatternDefByName(
     d_techFileId
     t_name
     )
     => d_widthSpacingSnapPatternDefId / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Searches for the widthSpacingSnapPatternDef object with the specified name in the specified technology file or a referenced technology database in an ITDB graph of the specified technology database.

Arguments

d_techFileId	Specifies the ID of the technology file in which the search is to be performed.
t_name	Specifies the object name.

Value Returned

d_widthSpacingSnapPatternDefId	ID of the `widthSpacingSnapPatternDef` object, if found.
`nil`	Returned in case of failure.

Example

```
techFindWidthSpacingSnapPatternDefByName(tech "wsspDef")
```

Returns `wsspDef` widthSpacingSnapPatternDef object found in the specified technology file.

Related Snap Patterns SKILL Functions

The related snap patterns SKILL functions are used to create, find, and access relatedSnapPatterns objects in the technology database.

techCreateRelatedSnapPatterns

```
techCreateRelatedSnapPatterns(
     d_techFileId
     t_name
     l_relatedSnapPatterns
     [l_extraLPP]
     [f_regionSnapPitchHorizontal]
     [f_regionSnapPitchVertical]
     )
     => d_techRelatedSnapPatternsId / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Creates a group of related snap patterns in the specified technology file.

Arguments

d_techFileId	Specifies the ID of the technology file in which the related snap patterns group is to be created.
t_name	Specifies the name for the related snap patterns group.

l_relatedSnapPatterns

Specifies the list of snap pattern definitions to include in the group.

l_relatedSnapPatterns:
list(*l_snapPatternDef*)

where,

- *l_snapPatternDef* is a snap pattern definition

  l_snapPatternDef:
  list(*t_snapPatternDefName*
  [l_widthSpacingPatternNames]
  [l_widthSpacingPatternGroupNames])

  - *t_snapPatternDefName* is the name for the snap pattern definition. This name can either refer to a snapPatternDef or a widthSpacingSnapPatternDef. If it refers to a snapPatternDef, the two optional arguments, l_widthSpacingPatternName and l_widthSpacingPatternGroupNames cannot be used.
  - *l_widthSpacingPatternNames* is a list of the width spacing pattern names allowed in this definition.

    l_widthSpacingPatternNames:
    list(*t_widthSpacingPatternName*)
  - *l_widthSpacingPatternGroupNames* is a list of the width spacing pattern group names allowed in the definition.

    l_widthSpacingPatternGroupNames:
    list
    (*t_widthSpacingPatternGroupName*)
  - *t_widthSpacingPatternGroupName* is a width spacing pattern group name allowed in the definition.

l_extraLPP

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

f_regionSnapPitchHorizontal

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

The default value is 0.

f_regionSnapPitchVertical

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

The default value is 0.

Value Returned

d_relatedSnapPatternsId

Returns the ID of the created related snap patterns group.

`nil` Returned in case of failure.

Example

```
techCreateRelatedSnapPatterns(tech
    "rsp1"
    '(
        ("snap1" nil ("wsp_group1"))
        ("snap2" "wsp1")
    )
    '("Active" "drawing")
    0.1
    0.2
    )
```

Creates a group of related snap patterns, `rsp1`, in the specified technology file.

techDeleteRelatedSnapPatterns

```
techDeleteRelatedSnapPatterns(
     d_techRSPId
     )
     => t / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Deletes the specified set of related snap patterns.

Arguments

d_techRSPId	Specifies the ID of the related snap patterns set to be deleted.

Value Returned

`t`	Specified related snap patterns set was deleted.
`nil`	Returned in case of failure.

Example

```
techDeleteRelatedSnapPatterns( rsp )
```

Deletes the related snap patterns set with ID `rsp`.

techFindRelatedSnapPatterns

```
techFindRelatedSnapPatterns(
     d_techFileId
     t_name
     )
     => d_RelatedSnapPatternsId / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Searches for the related snap patterns group with the specified name in the specified technology file or a referenced technology database in an ITDB graph of the specified technology database.

Arguments

d_techFileId	Specifies the ID of the technology file in which the search is to be performed.
t_name	Specifies the name of the related snap patterns group.

Value Returned

l_relatedSnapPatternIds	Returns the ID of the related snap patterns group.
nil	Returned in case of failure.

Example

```
techFindRelatedSnapPatterns(tech "rsp1")
```

Returns the ID of `rsp1` related snap pattern group found in the specified technology file.

techGetRelatedSnapPatterns

```
techGetRelatedSnapPatterns(
     d_techFileId
     )
     => l_relatedSnapPatternIds / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Returns a list of all related snap pattern IDs in the specified technology file.

Arguments

d_techFileId	Specifies the ID of the technology file in which the listing is performed.

Value Returned

l_relatedSnapPatternIds	Returns a list of related snap pattern IDs.
nil	Returned in case of failure.

Example

```
techGetRelatedSnapPatterns( tech )
```

Returns a list of all related snap pattern IDs found in the specified technology file.

15

Trim Layer Functions

The "trim layer" SKILL functions retrieve information about the trim layers and the metal and poly layers trimmed by these trim layers.

techGetTrimLayer

```
techGetTrimLayer(
     d_techFileID
     tx_layer
     [ t_color t_colorState ]
     )
     => l_trimLayer / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Returns the trim layer that trims *tx_layer* (metal or poly layer) with the specified color and color state combination. If only *d_techFileID* and *tx_layer* are specified, it returns a list of trim layers that trim *tx_layer* for all valid color and color state combinations.

Arguments

d_techFileID	The database identifier of the technology file.
tx_layer	The metal, poly, or local interconnect layer name or number.
t_color	The mask color. Valid values: `grayColor`, `mask1Color`, `mask2Color`, `mask3Color`
t_colorState	The color state. Valid values: `locked`, `unlocked`

Value Returned

l_trimLayer	The trim layer or a list of trim layers that trim the specified metal or poly layer.
`nil`	No trim layer trims the specified metal or poly layer, or an error condition occurred, such as an invalid input value.

Examples

```
techGetTrimLayer(tfid "Metal1")
=>((("mask1Color" "locked")
    ("trimMetal1" "mask1Color" "locked")
   )
   (("mask1Color" "unlocked")
    ("trimMetal1" "mask1Color" "unlocked")
   )
   (("mask2Color" "locked")
    ("trimMetal2" "mask2Color" "locked")
   )
   (("mask2Color" "unlocked")
    ("trimMetal2" "mask2Color" "unlocked")
   )
  )
```

Returns a list of trim layers that trim metal layer `Metal1`.

```
techGetTrimLayer(techfileId "Metal1" "mask2Color" "locked")
=>("trimMetal2" "mask2Color" "locked")
```

Returns the trim layer that trims layer `Metal1` with color `mask2Color` and color state `locked`.

techGetTrimLayers

```
techGetTrimLayers(
     d_techFileID
     tx_layer
     [ t_color t_colorState ]
     )
     => l_trimLayers / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Returns a list of the trim layers that trim *tx_layer* (metal or poly layer) with the specified color and color state combination. If only *d_techFileID* and *tx_layer* are specified, it returns a list of the trim layers that trim *tx_layer* for all valid color and color state combinations. If all arguments are specified, it returns a list of the trim layers that trim *tx_layer* for the specified color and color state combinations.

Arguments

d_techFileID	The database identifier of the technology file.
tx_layer	The metal, poly, or local interconnect layer name or number.
t_color	The mask color.
	Valid values: `grayColor`, `mask1Color`, `mask2Color`, `mask3Color`
t_colorState	The color state.
	Valid values: `locked`, `unlocked`

Value Returned

l_trimLayers	The trim layer or layers that trim the specified metal or poly layer, along with their color and color state.
`nil`	No trim layer trims the specified metal or poly layer, or an error condition occurred, such as an invalid input value.

Examples

```
techGetTrimLayers(tfid "Metal1")
=>((("mask1Color" "locked")
    ("trimMetal1" "mask1Color" "locked")
   )
   (("mask1Color" "unlocked")
    ("trimMetal1" "mask1Color" "unlocked")
   )
   (("mask2Color" "locked")
    ("trimMetal2" "mask2Color" "locked")
   )
   (("mask2Color" "unlocked")
    ("trimMetal2" "mask2Color" "unlocked")
   )
  )
```

Returns a list of the trim layers that trim metal layer `Metal1`.

```
techGetTrimLayers(techfileId "Metal1" "mask2Color" "locked")
=>("trimMetal2" "mask2Color" "locked") ("trimMetal3" "mask2Color" "any")
```

Returns a list of the trim layers that trim metal layer `Metal1` with color `mask2Color` and color state `locked`.

techGetTrimmedLayers

```
techGetTrimmedLayers(
     d_techFileID
     tx_layer
     [ t_color t_colorState ]
     )
     => l_trimmedLayers / nil
```

Description

(Virtuoso Advanced Node for Layout Only) Returns a list of metal or poly layers trimmed by *tx_layer* with the specified color and color state combination. If only *d_techFileID* and *tx_layer* are specified, it returns a list of metal or poly layers trimmed by *tx_layer* for all valid color and color state combinations.

Arguments

d_techFileID	The database identifier of the technology file.
tx_layer	The trim layer name or number.
t_color	The mask color.
	Valid values: `grayColor`, `mask1Color`, `mask2Color`, `mask3Color`
t_colorState	The color state.
	Valid values: `locked`, `unlocked`

Value Returned

l_trimmedLayers	A list of metal or poly layers trimmed by the specified trim layer.
`nil`	No metal or poly layers are trimmed by the specified trim layer, or an error condition occurred, such as an invalid input value.

Examples

```
techGetTrimmedLayers(tfid "trimMetal")
=>((("mask1Color" "locked")
    (("Metal1" "mask1Color" "locked"))
   )
   (("mask1Color" "unlocked")
    (("Metal1" "mask1Color" "unlocked"))
   )
   (("mask2Color" "locked")
    (("Metal2" "mask2Color" "locked") ("Metal1" "mask2Color" "any"))
   )
  )
```

Returns a list of metal layers trimmed by trim layer `trimMetal`.

```
techGetTrimmedLayers(tfid "trimMetal" "mask2Color" "locked")
=>(("Metal2" "mask2Color" "locked") ("Metal1" "mask2Color" "any"))
```

Returns a list of metal layers trimmed by trim layer `trimMetal` with color `mask2Color` and color state `locked`: trim layer `trimMetal` with `mask2Color` and color state `locked` trims `Metal2` with `mask2Color` and color state `locked` and `Metal1` with `mask2Color` irrespective of its color state.

16

Wire Profile and Finger Definition Functions

The wire profiles and finger definition SKILL functions are used to create and work with wire profiles and finger definitions.

techCreateFingerDef

```
techCreateFingerDef(
     d_techFileID
     t_name
     l_typeSpec
     )
     => d_fdID / nil
```

Description

(Virtuoso MultiTech Framework) Creates a finger definition in the specified technology file.

Arguments

d_techFileID	The database identifier of the technology file.
t_name	The name of the finger definition.

l_typeSpec

A list indicating the type and related details. The list format varies depending on the type, which can be one of these:

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

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

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

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

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

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

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

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

Value Returned

d_fdID The ID of the newly created finger definition.

`nil` A finger definition was not created.

Example

```
techCreateFingerDef(
    tf
    "fdName"
    '(type "oblong"
        length 0.5
        width 0.2
        layer "metal1"
    )
)
```

Creates a finger definition named `fdName` in the `tf` technology file.

techCreateWireProfile

```
techCreateWireProfile(
     d_techFileID
     t_name
     t_direction
     l_wpPoints
     [ g_fromVendor ]
     [ x_diameter ]
     [ t_material ]
     [ n_color ]
     [ g_visibility ]
     )
     => d_wireProfileID / nil
```

Description

(Virtuoso MultiTech Framework) Creates a wire profile object in the specified technology file.

Arguments

d_techFileID	The database identifier of the technology file.
t_name	The name of the wire profile.
t_direction	The direction of the wire profile. Valid values: `forward`, `reverse`

l_wpPoints	A list of points of the type `switch` or `point`.
	The list has the following syntax:
	`list( { l_hvPoint | l_switchPoint } … )`
	These arguments are described as follows:
	■ *l_hvPoint*: A list with the following syntax:
	`list ( type "point" horizontal l_point vertical l_point [horizontalTurn f_hTurn] )`

Here:

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

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

  Here:

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

- *f_hTurn*: Specifies a horizontal angle in the wire. Valid values: a float number in the range `-90` to `+90`. Default value: `0`

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

```
list(
     type "switch"
    )
```

g_fromVendor	Indicates whether the wire profile is from a vendor. Boolean value: `nil` (default), `t`
x_diameter	The diameter of the wire profile. Default value: `0`
t_material	The wire profile material name as a string. Default value: an empty string
n_color	The color index of the wire profile. Default value: `0`
g_visibility	The visibility of the wire profile. Boolean value: `nil` (default), `t`

Value Returned

d_wireProfileID	The ID of the newly created wire profile.
`nil`	A wire profile was not created.

Example

```
techCreateWireProfile(
    tf
    "wpName"
    "forward"
    '(
        (type "switch")
        (type "point")
        horizontal (type "percent"  value 5.0  locked t)
        vertical (type "length"  value 3.2  max 2.5  min 0.2  step 1.0)
        horizontalTurn 2.5
        )
        (type "switch")
    )
    ?fromVendor t
```

```
    ?diameter 3.25
    ?material "Gold"
)
```

Creates a wire profile named `wpName` in the `forward` direction. It has an entry each for a switch and a point, and the horizontal angle is set to `2.5`. It is from a vendor. The diameter is `3.25` and the material is `Gold`.

techCreateWireProfileGroup

```
techCreateWireProfileGroup(
     d_techFileID
     t_name
     l_wireProfileNames
     [?vendor t_vendor ]
     [?logo t_logoFile ]
     )
     => d_wireProfileGroupID / nil
```

Description

(Virtuoso MultiTech Framework) Creates a wire profile group object in the specified technology file.

Arguments

d_techFileID	The database identifier of the technology file.
t_name	The name of the wire profile group.
l_wireProfileNames	A list of wire profile names to be included in the group.
t_vendor	The name of the vendor as a string. Default value: an empty string
t_logoFile	The logo filename as a string. Default value: an empty string

Value Returned

d_wireProfileGroupID	The ID of the newly created wire profile group.
nil	A wire profile group was not created.

Example

```
techCreateWireProfileGroup(tf
    "wpgName" list("wpA" "wpB" "wpC" "wpD")
    ?logo "wpgLogo"
    )
```

Creates a wire profile group named `wpgName` including the wire profiles `wpA`, `wpB`, `wpC`, and `wpD`. The logo file is `wpgLogo`. A vendor is not specified.

techDeleteFingerDef

```
techDeleteFingerDef(
     d_fdID
     )
     => t / nil
```

Description

(Virtuoso MultiTech Framework) Deletes a specified finger definition.

Arguments

d_fdID	The database identifier of the finger definition to be deleted.

Value Returned

`t`	The specified finger definition has been deleted.
`nil`	The specified finger definition could not be deleted.

Example

```
techDeleteFinferDef(fdID)
```

Deletes the finger definition with the ID `fdID`.

techDeleteWireProfile

```
techDeleteWireProfile(
     d_wpID
     )
     => t / nil
```

Description

(Virtuoso MultiTech Framework) Deletes a specified wire profile.

Arguments

d_wpID	The database identifier of the wire profile object to be deleted.

Value Returned

t	The specified wire profile has been deleted.
nil	The specified wire profile could not be deleted.

Example

```
techDeleteWireProfile(wpID)
```

Deletes the wire profile with the ID `wpID`.

techDeleteWireProfileGroup

```
techDeleteWireProfileGroup(
     d_wpgID
     )
     => t / nil
```

Description

(Virtuoso MultiTech Framework) Deletes a specified wire profile group.

Arguments

d_wpgID	The database identifier of the wire profile group to be deleted.

Value Returned

`t`	The specified wire profile group has been deleted.
`nil`	The specified wire profile group could not be deleted.

Example

```
techDeleteWireProfileGroup(wpgID)
```

Deletes the wire profile group with the ID `wpgID`.

techFindFingerDef

```
techFindFingerDef(
     d_techFileID
     t_name
     )
     => d_fdID / nil
```

Description

(Virtuoso MultiTech Framework) Returns the ID of a specified finger definition, if located in a specified technology file.

Arguments

d_techFileID	The database identifier of the technology file.
t_name	The name of the finger definition.

Value Returned

d_fdID	The ID of the specified finger definition, if located.
nil	The specified finger definition could not be located.

Example

```
techFindFingerDef(tf "fd1")
```

Returns the ID of a finger definition with the name `fd1` in the `tf` technology file.

techFindWireProfile

```
techFindWireProfile(
     d_techFileID
     t_name
     )
     => d_wireProfileID / nil
```

Description

(Virtuoso MultiTech Framework) Returns the ID of a specified wire profile name, if located in a specified technology file.

Arguments

d_techFileID	The database identifier of the technology file.
t_name	The name of the wire profile.

Value Returned

d_wireProfileID	The ID of the specified wire profile, if located.
nil	The specified wire profile could not be located.

Example

```
techFindWireProfile(tf "wp1")
```

Locates a wire profile with the name `wp1`.

techFindWireProfileGroup

```
techFindWireProfileGroup(
     d_techFileID
     t_name
     )
     => d_wireProfileGroupID / nil
```

Description

(Virtuoso MultiTech Framework) Returns the ID of a specified wire profile group name, if located in a specified technology file.

Arguments

d_techFileID	The database identifier of the technology file.
t_name	The name of the wire profile group.

Value Returned

d_wireProfileGroupID	The ID of the specified wire profile group, if located.
nil	The specified wire profile group could not be located.

Example

```
techFindWireProfileGroup(tf "wpg1")
```

Locates a wire profile group with the name `wpg1`.

techImportWireProfileSet

```
techImportWireProfileSet(
     d_techID
     x_fileName
     t_groupName
     )
     => t / nil
```

Description

(Virtuoso MultiTech Framework) Imports wire profile data from an XML file into a technology database. If a profile already exists in the technology database, it is overwritten with the profile specified in the XML file.

Note: The XML file might contain length values in other units but they are converted and stored as microns (µm).

Arguments

d_techID	The database identifier of the technology database into which to import the data.
x_fileName	The path and name of the XML file from which wire profile data is to be imported.
t_groupName	The wire profile group to which all profiles imported from the XML file are added.

Value Returned

`t`	Wire profile data was successfully imported.
`nil`	The specified technology database or XML file does not exist.

Example

```
techImportWireProfileSet(tf "wpg1.xml" "wpg1")
=> t
```

Imports wire profile data from `wpg1.xml` into the `wpg1` wire profile group.

techExportWireProfileSet

```
techExportWireProfileSet(
     d_techID
     x_fileName
     [t_groupName]
     )
     => t / nil
```

Description

(Virtuoso MultiTech Framework) Exports wire profile data from a technology database into an XML file. You can export either all profiles in the technology database or only those that are in a particular group. Default values are not exported.

Arguments

d_techID	The database identifier of the technology database from which to export the data.
x_fileName	The name of the XML file into which wire profile data is to be exported.
t_groupName	A wire profile group. If specified, only the profiles listed in the group are exported. If not specified, all profiles in the technology database are exported.

Value Returned

`t`	Wire profile data was successfully exported.
`nil`	The specified technology database or XML file does not exist.

Example

```
techExportWireProfileSet(tf "wpg1.xml" "wpg1")
=> t
```

Exports wire profile data from the `wpg1` profile group into the `wpg1.xml` file.

A

Accessing Technology Databases with the ~> Operator

You can use the access (`~>`) operator to retrieve information from a technology database. If the requested data exists in the technology database, the operator returns an attribute or a property; otherwise, it returns `nil`. You can use the access operator to both set and retrieve the value of an attribute or a property.

Note: You cannot modify via definitions, via specifications, and site definitions with SKILL. To modify a definition, you must delete the existing definition and re-create it.

techID~>

```
techID~>
     layers
     derivedLayers
     purposes
     purposeDefs
     groups
     lps
     maxPriority
     createTime
     timeStamp
     timeStamps
     libName
     fileName
     path
     mode
     needRefresh
     constraintGroups
     viaDefs
     viaVariants
     siteDefs
     viaSpecs
     modifiedButNotSaved
     refs
     refLibNames
     usedIn
     hasConflict
     allRefs
     allRefLibNames
     distanceMeasure
     processFamily
     snapPatternDefs
     widthSpacingSnapPatternDefs
```

Description

Retrieves the database identifiers or values for the specified attribute from the technology database identified by *techID*.

Attributes

layers	A list of database identifiers of all the layers found in the technology database.
derivedLayers	A list of database identifiers of all the derived layers found in the technology database.

`purposes`	A list of all the purpose names found in the technology database.
`purposeDefs`	A list of database identifiers of all the purposes found in the technology database.
`groups`	A list of database identifiers for the groups found in the technology database.
`lps`	A list of database identifiers of all the layer-purpose pairs found in the technology database.
`maxPriority`	The maximum priority found in the technology database for a layer-purpose pair.
`needRefresh`	Indicates whether the technology database is synchronized with the database on disk. Returns `t` if any changes have been made in the memory after the technology database was last loaded.
`createTime`	The date and time of creation of the technology database.
`timeStamp`	The date and time when the technology database was last modified.

`timeStamps`	The time stamp object for the technology database. The following returns integer counters for the data types defined in the technology database loaded in virtual memory:
	techID`~>timeStamps~>`
	`techDataType`
	`propDataType`
	`groupDataType`
	`groupMemDataType`
	`layerDataType`
	`purposeDataType`
	`siteDefDataType`
	`viaDefDataType`
	A counter is updated each time you modify a data type.

`libName`	The name of the technology database identified by *techID*.
`fileName`	The name of the binary technology database identified by *techID*. The name of the binary technology database is always `tech.db`.
`path`	The path where the binary technology database identified by *techID* is stored on the disk.

`mode`	The mode in which a binary technology database is loaded in virtual memory. The three valid modes are `r` (read), `w` (write), and `a` (append).
`constraintGroups`	A list of database identifiers for all the constraint groups found in the technology database.
`viaDefs`	A list of database identifiers for all the vias found in the technology database.
`viaVariants`	A list of database identifiers for all the via variants found in the technology database.
`siteDefs`	A list of database identifiers for all the site definitions found in the technology database.
`viaSpecs`	The database identifier of the array of vias and via variants found in the technology database.
`modifiedButNotSaved`	`TRUE` or `FALSE` depending on whether the modifications made to the technology data loaded in virtual memory have been saved.
`refs`	A list of database identifiers of all technology libraries referenced by the technology database.
`refLibNames`	A list of names of the technology libraries referenced by the technology database.
`usedIn`	The database identifier of the technology library that references the technology database identified by *`techID`*.
`hasConflict`	`t` if the technology database contains a conflict; `nil` if no conflict exists.
`allRefs`	The database identifiers of all the technology libraries referenced to build the incremental technology database identified by *`techID`*.
`allRefLibNames`	The names of all the technology libraries referenced to build the incremental technology database identified by *`techID`*.
`distanceMeasure`	The method specified in the technology library for measuring spacing, `euclidian` or `manhattan`.
`processFamily`	The process family name specified in the technology database.

`snapPatternDefs` (Virtuoso Advanced Node for Layout Standard) A list of database identifiers of all the snap pattern definitions found in the technology database.

`widthSpacingSnapPatternDefs`

(Virtuoso Advanced Node for Layout Only) A list of database identifiers of all the width spacing snap pattern definitions found in the technology database.

Examples

```
LIB="myLib"
tfid=techGetTechFile(ddGetObj(LIB))
```

Stores in *tfid* the database identifier of the technology library associated with design library *myLib*.

```
tfid~>usedIn
=> (db:0x1742ea1a)
```

Retrieves the database identifier of the technology library that references the technology library identified by *tfid*.

```
tfid~>createTime
=> "Nov  5 03:59:17 2015"
```

Displays the date and time of creation of the technology library.

```
tfid~>widthSpacingSnapPatternDefs
=> (db:0x1742c61a db:0x1742c61b db:0x1742c61c db:0x1742c61d)
```

Retrieves the database identifiers of the width spacing snap pattern definitions.

```
techID~>refs=list(tech1ID …)
```

Replaces the ordered list of the referenced technology libraries in the specified technology library with the specified list.

techID~>layers

```
techID~>layers~>
     number
     name
     abbrev
     lps
     incompatibleLayerNames
     backside
     material
     valid
     allowSetToValid
     allowSetToValidInSession
```

Description

Retrieves layer information from the technology database identified by *techID*.

Attributes

number	A list of numbers assigned to layers.
name	A list of layer names.
abbrev	A list of abbreviated names assigned to layers.
lps	A list of database identifiers for layer-purpose pairs.
incompatibleLayerNames	A list of names of all incompatible layers.
backside	A flag corresponding to each layer, t or nil.
material	A list of layer functions, such as metal, cut, poly, nwell, and pwell.
valid	A flag corresponding to each layer. t if the layer is valid; nil if the layer is invalid.
allowSetToValid	A flag corresponding to each layer. t if the layer can be set as valid; nil if the layer cannot be set as valid.
allowSetToValidInSession	A flag corresponding to each layer. t if the layer can be set as valid during a session; nil if the layer cannot be set as valid during a session.

Examples

```
tfid~>layers
=> (db:0x1742e89a db:0x1742e89b db:0x1742e89c db:0x1742e89d db:0x1742e89e
    db:0x1742e89f db:0x1742e8a0 db:0x1742e8a1 db:0x1742e8a2 db:0x1742e8a3
    )


tfid~>layers~>name
=> ("Via1" "Metal2" "MIMBOT" "MIMTOP" "Via2"
    "Metal3" "Via3" "Metal4" "Via4" "Metal5
    )


tfid~>layers~>material
=> ("cut" "metal" "mimcap" "mimcap" "cut"
    "metal" "cut" "metal" "cut" "metal"
    )


tfid~>layers~>allowSetToValid
=> (t t t t t
    t t t t t
    )
```

techID~>derivedLayers

```
techID~>derivedLayers~>
     number
     name
     layer1
     layer1Num
     layer2
     layer2Num
     op
     params
```

Description

Retrieves derived layer information from the technology database identified by *techID*.

Attributes

number	A list of numbers assigned to derived layers
name	A list of derived layer names
layer1	A list of database identifiers identifying the first layer in the derived layers
layer1Num	A list of layer numbers identifying the first layer in the derived layers
layer2	A list of database identifiers identifying the second layer in the derived layers
layer2Num	A list of layer numbers identifying the second layer in the derived layers
op	A list of logical operators used to create the derived layers
params	A list of derived layer parameters

Examples

```
tfid~>derivedLayers
=> (db:0x1742e99a db:0x1742e99b db:0x1742e99c db:0x1742e99d db:0x1742e7aa
    db:0x1742e7ab db:0x1742e7ac db:0x1742e7ad db:0x1742e7ae db:0x1742e7af
    )
```

```
tfid~>derivedLayers~>name
=> ("M2WSP" "M3WSP" "M4WSP" "M5WSP" "bulkActive"
    "Gate" "PolyInterConn" "SD" "cutSubstrate" "Implant"
    )


tfid~>derivedLayers~>layer1~>name
=> ("Metal2" "Metal3" "Metal4" "Metal5" "Active"
    "Poly" "Poly" "Li1" "substrate" "NImplant"
    )


tfid~>derivedLayers~>layer2~>name
=> (nil nil nil nil nil
    "Active" "CutPoly" "bulkActive" "NWell" "PImplant"
    )


tfid~>derivedLayers~>op
=> ("select" "select" "select" "select" "select"
    "and" "not" "inside" "not" "or"
    )


tfid~>derivedLayers~>params
=> (("selectShapesWithPurpose" "localWSP")
    ("selectShapesWithPurpose" "localWSP")
    ("selectShapesWithPurpose" "localWSP")
    ("selectShapesWithPurpose" "localWSP")
    ("selectShapesWithPurpose" "drawing")
    nil nil nil nil nil
    )
```

techID~>purposeDefs

```
techID~>purposeDefs~>
     number
     name
     abbrev
     isReserved
     voltageRange
     parent
     sigType
     description
     valid
     allowSetToValid
     allowSetToValidInSession
```

Description

Retrieves purpose information from the technology database identified by *techID*.

Attributes

number	A list of numbers assigned to purposes.
name	A list of purpose names.
abbrev	A list of abbreviated names assigned to purposes.
isReserved	A flag corresponding to each purpose. t if the purpose is reserved; nil if the purpose is not reserved.
voltageRange	A list of minimum-maximum voltage values.
parent	A list of database identifiers identifying the parent purpose for each purpose found in the technology database.
sigType	A list of signal types.
description	A list of descriptions.
valid	A flag corresponding to each purpose. t if the purpose is valid; nil if the purpose is invalid.
allowSetToValid	A flag corresponding to each purpose. t if the purpose can be set as valid; nil if the purpose cannot be set as valid.

`allowSetToValidInSession`

A flag corresponding to each purpose. `t` if the purpose can be set as valid during a session; `nil` if the purpose cannot be set as valid during a session.

Examples

```
tfid~>purposeDefs
=> (db:0x1742cc9a db:0x1742cc9b db:0x1742cc9c db:0x1742cc9d db:0x1742cc9e
    db:0x1742cc9f db:0x1742cca0 db:0x1742cca1 db:0x1742cca4 db:0x1742cca5
    db:0x1742e226 db:0x1742e227 db:0x1742e228 db:0x1742e229 db:0x1742e22a
    )


tfid~>purposeDefs~>number
=> (-1 -2 -3 -4 -5
    -6 -7 -8 -11 -12
    1000 10001 10002 10003 10004
    )


tfid~>purposeDefs~>name
=> ("drawing" "fill" "slot" "OPCSerif" "OPCAntiSerif"
    "annotation" "gapFill" "redundant" "fillOPC" "customFill"
    "dummy" "0p9" "1p2" "1p5" "1p8"
    )


tfid~>purposeDefs~>isReserved
=> (t t t t t
    t t t t t
    nil nil nil nil nil
    )


tfid~>purposeDefs~>voltageRange
=> (nil nil nil nil nil
    nil nil nil nil nil
    nil (0.0 0.9) (0.0 1.2) (0.0 1.5) (0.0 1.8)
    )
```

techID~>lps

```
techID~>lps~>
     number
     name
     layer
     purpose
     purposeDef
     packet
     priority
     changeLayer
     dragEnable
     selectable
     valid
     visible
     allowSetToValid
     allowSetToValidInSession
     tech
     techFile
```

Description

Retrieves layer-purpose pair information from the technology database identified by *techID*.

Attributes

number	A list of layer numbers identifying various layer-purpose pairs.
name	A list of layer names identifying various layer-purpose pairs.
layer	A list of database identifiers identifying the layers in layer-purpose pairs.
purpose	A list of purpose names identifying the purposes in layer-purpose pairs.
purposeDef	A list of database identifiers identifying the purposes in layer-purpose pairs.
packet	A list of names identifying the packets assigned to the layer-purpose pairs.
priority	A list of integers identifying the priorities assigned to layer-purpose pairs.
changeLayer	A flag corresponding to each layer-purpose pair. t if the layer contributes to changedLayer when a DRC check is run; nil if the layer does not contribute to changedLayer.

`dragEnable`	A flag corresponding to each layer-purpose pair. `t` if a shape created on the layer can be dragged; `nil` if a shape created on the layer cannot be dragged.
`selectable`	A flag corresponding to each layer-purpose pair. `t` if the layer-purpose pair is selectable; `nil` if the layer-purpose pair is not selectable.
`valid`	A flag corresponding to each layer-purpose pair. `t` if the layer-purpose pair is valid; `nil` if the layer-purpose pair is invalid.
`visible`	A flag corresponding to each layer-purpose pair. `t` if the layer-purpose pair is visible; `nil` if the layer-purpose pair is invisible.
`allowSetToValid`	A flag corresponding to each layer-purpose pair. `t` if the layer-purpose pair can be set as valid; `nil` if the layer-purpose pair cannot be set as valid.
`allowSetToValidInSession`	A flag corresponding to each layer-purpose pair. `t` if the layer-purpose pair can be set as valid during a session; `nil` if the layer-purpose pair cannot be set as valid during a session.
`tech`	A list of database identifiers identifying the technology databases in which the layer-purpose pairs are defined.
`techFile`	A list of database identifiers identifying the technology databases in which the layer-purpose pairs are defined.

Examples

```
tfid~>lps
=> (db:0x1b2b8e30 db:0x1b2cb110 db:0x1b2bbde0 db:0x1b2c88b0 db:0x1b2cb310
    db:0x1b2cae90 db:0x1b2cafb0 db:0x1b2ca170 db:0x1b2ca280 db:0x1b2ca3a0
    )


tfid~>lps~>packet
=> ("background" "grid" "grid1" "annotate" "annotate1"
    "annotate2" "annotate3" "annotate4" "annotate5" "annotate6"
    )


tfid~>lps~>dragEnable
=> (nil nil nil t t
    t t t t t
    )
```

techID~>viaDefs

```
techID~>viaDefs~>
     objType
     tech
     techFile
     name
     layer1
     layer1Num
     layer2
     layer2Num
     resistancePerCut
     implant1
     implant2
     wellOrSubstrate
     params
     libName
     cellName
     viewName
```

Description

Retrieves information related to via definitions from the technology database identified by *techID*.

Attributes

objType	A list of via definition types, stdViaDef or customViaDef.
tech	A list of database identifiers identifying the technology databases in which the vias are defined.
techFile	A list of database identifiers identifying the technology databases in which the vias are defined.
name	A list of via names.
layer1	A list of database identifiers identifying the first layer in each via.
layer1Num	A list of layer numbers identifying the first layer in each via.
layer2	A list of database identifiers identifying the second layer in each via.
layer2Num	A list of layer numbers identifying the second layer in each via.
resistancePerCut	A list of resistance values assigned per cut.

implant1	(stdViaDefs) A list of database identifiers identifying the first implant layer of the vias.
implant2	(stdViaDefs) A list of database identifiers identifying the second implant layer of the vias.
wellOrSubstrate	(stdViaDefs) A list of database identifiers identifying the well and substrate layers of the vias.
params	(stdViaDefs) A list of values defining the geometry of the vias.
libName	(customViaDefs) A list of design libraries that contain the custom via cellviews.
cellName	(customViaDefs) A list of cell names for the custom vias.
viewName	(customViaDefs) A list of view names for the custom vias.

Examples

```
tfid~>viaDefs
=> (db:0x1742c39a db:0x1742c39b db:0x1742c39c db:0x1742c39d db:0x1742c39e
    db:0x1742c39f db:0x1742c3a0 db:0x1742c3a1 db:0x1742c3a2 db:0x1742c3a3
    )


tfid~>viaDefs~>objType
=> ("stdViaDef" "stdViaDef" "stdViaDef" "stdViaDef" "stdViaDef"
    "stdViaDef" "stdViaDef" "stdViaDef" "stdViaDef" "customViaDef"
    )


tfid~>viaDefs~>name
=> ("M2_MIMTOP" "M2_MIMBOT" "M2M1_stdV" "M3M2_stdV" "M4M3_stdV"
    "M5M4_stdV" "M6M5_stdV" "M7M6_stdV" "M8M7_stdV" "M2_M1"
    )


tfid~>viaDefs~>layer1~>name
=> ("Metal3" "Metal3" "Metal1" "Metal2" "Metal3"
    "Metal4" "Metal5" "Metal6" "Metal7" "Metal1"
)


tfid~>viaDefs~>layer2~>name
=> ("MIMTOP" "MIMBOT" "Metal2" "Metal3" "Metal4"
    "Metal5" "Metal6" "Metal7" "Metal8" "Metal2"
    )
```

```
tfid~>viaDefs~>libName
=> (nil nil nil nil nil
    nil nil nil nil "cdn20FF"
    )


tfid~>viaDefs~>params
=> (("Via2" 0.032 0.032 1 1
     (0.032 0.032)
     (0.0 0.0)
     (0.01 0.01)
     (0.0 0.0)
     (0.0 0.0)
     (0.0 0.0) nil nil
     )
    ("Via1" 0.032 0.032 1 1
     (0.032 0.032)
     (0.0 0.0)
     (0.01 0.01)
     (0.0 0.0)
     (0.0 0.0)
     (0.0 0.0) nil nil
     )
    ("Via3" 0.032 0.032 1 1
     (0.032 0.032)
     (0.0 0.0)
     (0.01 0.01)
     (0.0 0.0)
     (0.0 0.0)
     (0.0 0.0) nil nil
     )
    ("Li1" 0.03 0.082 1 1
     (0.2 0.2)
     (-0.01 0.038)
     (0.0 0.0)
     (0.0 0.0)
     (0.0 0.0)
     (0.0 0.0)
     (0.045 0.065)
     (0.065 0.065)
     )
```

techID~>viaVariants

```
techID~>viaVariants~>
     objType
     cellView
     tech
     name
     params
     viaDef
     viaDefName
```

Description

Retrieves information related to via variants from the technology database identified by *techID*.

Attributes

`objType`	A list of object types, `stdViaVariant` or `customViaVariant`.
`cellView`	A list of cellviews in which the via variants are defined. Via variants can be defined in the technology file or in a cellview. If a via variant is defined in the technology file, the `cellView` attribute for it is `nil`.
`tech`	A list of database identifiers identifying the technology databases in which the via variants are defined.
`name`	A list of names of the via variants found in the technology database.
`params`	A list of parameters associated with the via variants.
`viaDef`	A list of database identifiers identifying the via definitions for which variants exist in the technology database.
`viaDefName`	A list of via definition names for which variants exist in the technology database.

Examples

```
tfid~>viaVariants~>objType
=> ("customViaVariant" "stdViaVariant")
```

```
tfid~>viaVariants~>name
=> ("vvtest3" "testStdVariant")

tfid~>viaVariants~>viaDefName
=> ("MineM2M1" "MineM3M2")
```

techID~>viaSpecs

```
techID~>viaSpecs~>
     tech
     techFile
     layer1
     layer1Num
     layer2
     layer2Num
     defaultViaDefs
     defaultViaDefNames
```

Description

Retrieves information related to via specifications from the technology database identified by *techID*.

Attributes

`tech`	A list of database identifiers identifying the technology databases in which the via specifications are defined.
`techFile`	A list of database identifiers identifying the technology databases in which the via specifications are defined.
`layer1`	A list of database identifiers identifying the first layer in each via specification.
`layer1Num`	A list of layer numbers identifying the first layer in each via specification.
`layer2`	A list of database identifiers identifying the second layer in each via specification.
`layer2Num`	A list of layer numbers identifying the second layer in each via specification.
`defaultViaDefs`	A list of database identifiers identifying the via definitions associated with the via specifications.
`defaultViaDefNames`	A list of names of the via definitions associated with the via specifications.

Examples

```
tfID~>viaSpecs
=> (db:0x16f4c81a db:0x16f4c81b db:0x16f4c81c db:0x16f4c81d db:0x16f4c81e
    db:0x16f4c81f db:0x16f4c820 db:0x16f4c821
    )


tfid~>viaSpecs~>layer1
=> (db:0x16f4d7aa db:0x16f4d7ac db:0x16f4d7b8 db:0x16f4d7b6 db:0x16f4d7b4
    db:0x16f4d7b2 db:0x16f4d7b0 db:0x16f4d7ae
    )


tfid~>viaSpecs~>layer2
=> (db:0x16f4d7a8 db:0x16f4d7aa db:0x16f4d7b6 db:0x16f4d7b4 db:0x16f4d7b2
    db:0x16f4d7b0 db:0x16f4d7ae db:0x16f4d7ac
    )


tfid~>viaSpecs~>defaultViaDefs
=> ((db:0x16f4e134)
    (db:0x16f4e134 db:0x16f4e133)
    (db:0x16f4e12d)
    (db:0x16f4e12e)
    (db:0x16f4e12f)
    (db:0x16f4e130)
    (db:0x16f4e131)
    (db:0x16f4e132)
    )


tfid~>viaSpecs~>defaultViaDefNames
=> (("M2_M1")
    ("M2_M1" "M3_M2")
    ("M9_M8")
    ("M8_M7")
    ("M7_M6")
    ("M6_M5")
    ("M5_M4")
    ("M4_M3")
    )


tfID=nth(0, tf~>viaSpecs)
=> db:0x0187090c
```

techID~>siteDefs

```
techID~>viaDefs~>
     objType
     tech
     techFile
     name
     type
     symmetricInX
     symmetricInY
     symmetricInR90
     width
     height
```

Description

Retrieves information related to site definitions from the technology database identified by *techID*.

Attributes

objType	A list of the types of site definitions, scalarSiteDefs or arraySiteDefs.
tech	A list of database identifiers identifying the technology databases in which the sites are defined.
techFile	A list of database identifiers identifying the technology databases in which the sites are defined.
name	A list of site definition names.
type	A list of site types, pad or core.
symmetricInX	A flag corresponding to each site. t if the site is symmetric in the X direction; nil if the site is not symmetric in the X direction.
symmetricInY	A flag corresponding to each site. t if the site is symmetric in the Y direction; nil if the site is not symmetric in the Y direction.
symmetricInR90	A flag corresponding to each site. t if the site is symmetric in rotation; nil if the site is not symmetric in rotation.
width	A list containing the width of each site.
height	A list containing the height of each site.

`sitePattern`	(`arraySiteDefs`) A list containing each scalar site definition in the array, along with its offset from the origin of the array and its orientation.

Examples

```
tfid~>siteDefs
=> (db:0x1742c41a)

tfid~>siteDefs~>objType
=> ("scalarSiteDef")

tfid~>siteDefs~>name
=> ("PP_90M2_12")

tfid~>siteDefs~>type
=> ("core")

tfid~>siteDefs~>symmetricInX
=> (t)

tfid~>siteDefs~>width
=> (0.09)

tf=techGetTechFile(ddGetObj("newTech18"))
=> db:0x011ea00e
site=car(tf~>siteDefs)
=> db:0x011ea40c
cv=geGetEditCellView()
=> db:0x011ea00d
row=dbCreateRow(cv site "myRow" '(0 0) 23)
=> db:0x011ea48c
row~>??
=> (db:0x011ea48c cellView db:0x011ea00d objType "row"
    prop nil groupMembers nil name
    "myRow" rowHeader db:0x011ea50c siteDef db:0x011ea40c
    siteOrient "R0" numSites 23 xy
    (0.0 0.0) orient "R0" markers nil
    )
```

```
row~>rowHeader~>??
=> (db:0x011ea50c cellView db:0x011ea00d objType "rowHeader"
    prop nil siteDef db:0x011ea40c siteDefName
    "DPHD1_site" siteDefWidth 574.84 siteDefHeight 1352.96
    bBox
    ((0.0 0.0)
    (13221.32 1352.96)
    ) rows
    (db:0x011ea48c)
row~>siteDef
=> db:0x011ea40c
row~>rowHeader~>siteDef
=> db:0x011ea40c
```

Creates a row, and then returns the site definition database identifier.

techID~>snapPatternDefs

```
techID~>snapPatternDefs~>
     name
     layer
     layerNum
     purpose
     purposeNum
     direction
     step
     snappingLayers
     type
     offset
     trackWidth
     trackGroups
```

Description

(Virtuoso Advanced Node for Layout Standard)(Virtuoso Advanced Node for Layout Standard) Retrieves information related to snap pattern definitions from the technology database identified by *techID*.

Attributes

`name`	A list of snap pattern definition names.
`layer`	A list of layers on which the snap pattern definitions apply.
`layerNum`	A list of layer numbers corresponding to these layers.
`purpose`	A list of purposes on which the snap pattern definitions apply.
`purposeNum`	A list of purpose numbers corresponding to these purposes.
`direction`	A list of direction values identifying the direction in which each snap pattern track is created.
`step`	A list of spacing values identifying the spacing between snap pattern tracks.
`snappingLayers`	A list of layers on which the shapes snap to the snap pattern.
`type`	A list of snap pattern definition types, `local` or `global`.
`offset`	A list of distance values defining for each snap pattern the distance of the first snap pattern track from the bottom or the left edge of the bounding box shape.

`trackWidth`	A list of widths of the physical shapes that each track represents.
`trackGroups`	(Virtuoso Advanced Node for Layout Only) A list of groups of tracks with variable spacing.

Examples

```
tfid~>snapPatternDefs
=> (db:0x1742c59a db:0x1742c59b db:0x1742c59c db:0x1742c59d db:0x1742c59e
    db:0x1742c59f db:0x1742c5a0 db:0x1742c5a1 db:0x1742c5a2 db:0x1742c5a3
    db:0x1742c5a4
    )


tfid~>snapPatternDefs~>name
=> ("GFG" "GPG86" "GPG90" "GPG94" "GPG102"
    "GPG104" "FB48" "fbd" "fb42" "fb44"
    "fb46"
)


tfid~>snapPatternDefs~>layer~>name
=> ("CellBoundary" "PPitch" "PPitch" "PPitch" "PPitch"
    "Poly" "FinArea" "FinArea" "FinArea" "FinArea"
    "FinArea"
    )


tfid~>snapPatternDefs~>snappingLayers
=> ((((layer "FinArea")
        (enclosures
            (0.007)
        )
        (purposes
            ("fin48")
        )
    )
    )
    (((layer "Poly")
        (enclosures
            (0.009)
        )
        (purposes
            ("drawing" "dummy")
        )
    )
    )
    ((layer "CutActive")
        (enclosures
            (0.024)
        )
    )
```

```
)
```

```
tfid~>snapPatternDefs~>type
=> ("global" "local" "local" "local" "local"
    "local" "local" "local" "local" "local"
    "local"
    )
```

techID~>widthSpacingSnapPatternDefs

```
techID~>widthSpacingSnapPatternDefs~>
     name
     layer
     layerNum
     purpose
     purposeNum
     period
     direction
     offset
     snappingLayers
     patternNames
     patternGroupNames
     defaultActiveName
```

Description

(Virtuoso Advanced Node for Layout Only) Retrieves information related to width spacing snap pattern definitions from the technology database identified by *techID*.

Attributes

name	A list of width spacing pattern definition names.
layer	A list of layers on which the width spacing snap pattern regions are drawn.
layerNum	A list of layer numbers corresponding to these layers.
purpose	A list of purposes to which the snap pattern definitions apply.
purposeNum	A list of purpose numbers corresponding to these purposes.
period	A list of spacing values for coarse-grain period tracks.
direction	A list of direction values identifying the direction in which period spacing is applied.
offset	A list of distance values defining the distance of the nearest period track to the anchor reference for each width spacing pattern.
snappingLayers	A list of layers to which the width spacing snap pattern definitions apply.
patternNames	A list of the allowed width spacing patterns.
patternGroupNames	A list of the allowed width spacing pattern groups.

`defaultActiveName`	A list of the default patterns visible in the layout in areas where no region has been drawn.

Examples

```
tfid~>widthSpacingSnapPatternDefs
=> (db:0x1742c61a db:0x1742c61b db:0x1742c61c db:0x1742c61d)

tfid~>widthSpacingSnapPatternDefs~>name
=> ("M2WSP" "M3WSP" "M4WSP" "M5WSP")

tfid~>widthSpacingSnapPatternDefs~>layer
=> (db:0x1742e89b db:0x1742e89f db:0x1742e8a1 db:0x1742e8a3)

tfid~>widthSpacingSnapPatternDefs~>layer~>name
=> ("Metal2" "Metal3" "Metal4" "Metal5")

tfid~>widthSpacingSnapPatternDefs~>purpose~>name
=> ("localWSP" "localWSP" "localWSP" "localWSP")

tfid~>widthSpacingSnapPatternDefs~>period
=> (0.768 0.768 0.768 0.768)

tfid~>widthSpacingSnapPatternDefs~>direction
=> ("vertical" "horizontal" "vertical" "horizontal")

tfid~>widthSpacingSnapPatternDefs~>snappingLayers
=> ((((layer "Metal2")))
    (((layer "Metal3")))
    (((layer "Metal4")))
    (((layer "Metal5")))
    )

tfid~>widthSpacingSnapPatternDefs~>patternNames
=> (("stdCell") nil nil nil)

tfid~>widthSpacingSnapPatternDefs~>defaultActiveName
=> ("minWidth" "minWidth" "minWidth" "minWidth")
```