cadence®

Virtuoso Studio Design Environment Adoption Guide

Product Version IC23.1
September 2023

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

Contents

1

Introduction to OpenAccess Adoption

The Virtuoso® Studio Design Environment Adoption provides the information needed to move an IC development environment, including C and SKILL code, from the CDB database to the Virtuoso Studio design environment on OpenAccess database.

With the help of this guide, you should be able to translate your data, convert your SKILL and C code, and successfully adopt OpenAccess.

Note: In past releases, the Virtuoso Studio design environment was called the Design Framework II (DFII) environment. As of IC6.1, the DFII name has changed to the Virtuoso Studio design environment to reflect the new environment and platform changes.

This guide includes information about the following areas:

- Technology file changes
- Behavioral changes
- Object support and creation
- Connectivity changes and enhancements
- C and SKILL function changes and new functions
- Development of the Virtuoso Studio design environment on OpenAccess
- CDB features not supported in OpenAccess
- Known conversion issues

This user guide is aimed at developers and designers of integrated circuits and assumes that you are familiar with:

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

- Component description format (CDF), which lets you create and describe your own components for use with Layout XL.

This chapter describes the relationships among the Virtuoso® Studio Design Environment on OpenAccess database components and how to convert data to Virtuoso Studio design environment.

Acronyms and Terminology

API	Application Programming Interface
CDBA	Cadence DataBase API. The C and SKILL API layer that is used to access the OpenAccess and CDB databases.
CDB	Cadence proprietary database
DFII	Design Framework II. The set of tools in this design environment include the Virtuoso physical design tools and the analog design/mixed signal tools. Customers customize the environment using the Cadence SKILL language. As of IC23.1, the Virtuoso Design Environment name has changed to Virtuoso Studio Design Environment to reflect the new environment and platform changes.
DDPI	Design Data Procedural Interface. The API layer that supports a variety of functionality including support of the 5.x library structure, file locking, data registry, library categories, limited design management capabilities (GDM), and startup file searching functionality.
OpenAccess	Application programming interface and reference database created as an interoperability platform for complex digital, analog and mixed-signal IC design.
Property Bag	Database files at the library, cell, and view levels that contain property information. OpenAccess is replacing property bags. Access to property information is currently made through CDBA `dd` functions.
Virtuoso Studio Design Environment on OpenAccess	System comprising the Virtuoso software tools, built on the OpenAccess database using the CDBA procedural interface.

What Is OpenAccess?

OpenAccess is an application programming interface (API) and reference database created as an interoperability platform for complex digital, analog, and mixed signal IC design. It is uniquely offered through a secure, open-community source program.

Both the suppliers and integrators of design automation tools benefit from the innovations in OpenAccess. OpenAccess provides electronic design suppliers with a versatile, high performance, high capacity database engineered for fast development. By using OpenAccess, electronic design suppliers can better focus on innovation while lowering development costs. When used as a common API and unified database, OpenAccess offers flow integrators an unprecedented ability to tailor distinctive flows using electronic design applications from multiple sources. Flows are integrated quickly and perform with greater efficiency.

OpenAccess is composed of various subsystems. The subsystems pertinent to the Virtuoso Studio design environment development are briefly described in the following sections. The infrastructure packages are designed with as few external dependencies as possible to minimize the number of packages that have to be published with OpenAccess.

For more information about OpenAccess, go to the following URLs.

http://www.si2.org

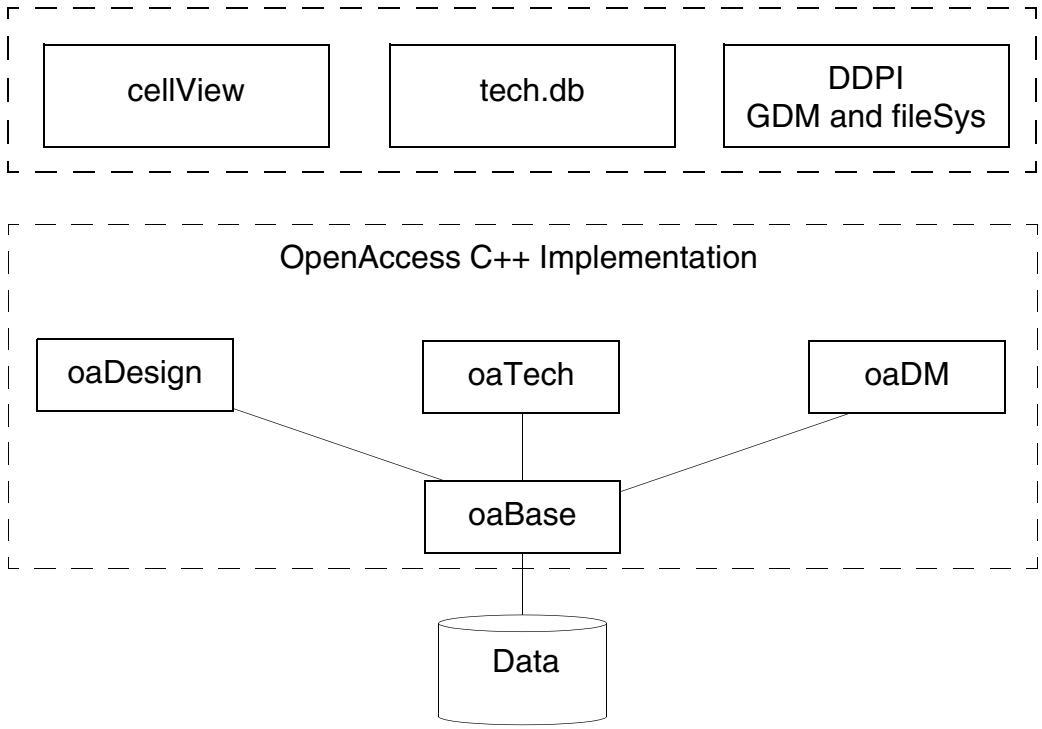
http://openeda.si2.org

Sponsored by Silicon Integration Initiative, Inc [Si2], OpenEDA.Si2.org is a restricted access site for the distribution of files from Si2 development groups (councils, projects, boards, or workgroups). Login accounts are required for file downloads and many site features.

Release notes are available which describe the product changes for each release of OpenAccess APIs.

The Database Infrastructure

The following diagram shows the relationships between the Virtuoso Studio design environment on OpenAccess components and the OpenAccess database components.



- oaDesign— This subsystem defines the objects that can exist in a design database.

  In OpenAccess, a design is just one of several types of cellviews. The OpenAccess `oaDesign` holds all the design data that describes a part of a design. It is a container for the connectivity, geometry, hierarchy, parasitics, and floorplanning information about a design.

  A Virtuoso database cellview ID (for example the `dbId` returned from a call to `dbOpenCellView()`) is the object associated with an `oaDesign`.

  A Virtuoso design data cellview ID (for example the `ddId` returned from a call to `ddGetObj()`) is the object associated with an `oaCellview`.

- `oaTech`—This subsystem defines the objects that support the OpenAccess technology database.

- `oaDM`—This subsystem defines design management.

  For more information, see Design Management on page 66.

Virtuoso Studio Design Environment on OpenAccess

The Virtuoso Studio design environment on OpenAccess is a system comprising the Virtuoso software tools, with access to the OpenAccess database through C and SKILL APIs.

The acronym CDB (Cadence Database) refers to the physical Cadence proprietary database, which previously was the only database that was supported for use with Virtuoso applications. The acronym CDBA (Cadence Database API) refers to the C and SKILL procedural interface extended to work on both CDB and OpenAccess. Previously implemented on Cadence proprietary technology, CDBA is now implemented on top of the OpenAccess database and has been significantly expanded to offer direct access to many new native OpenAccess objects.

Configuring OpenAccess

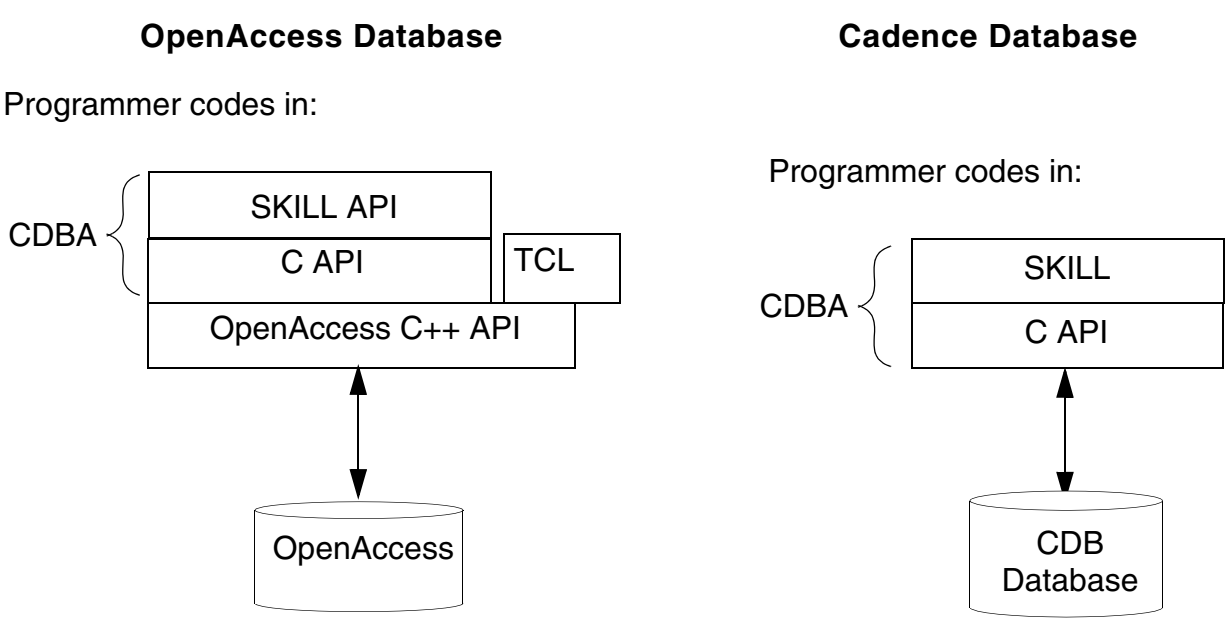
If the Cadence product you are using requires OpenAccess DM4, it will automatically install OpenAccess in the following directory:

```
<cadence_install_dir>/oa_v22.04.<xxx>
```

where `xxx` denotes the build number such as 001.

You can override the default OpenAccess installation by using the `OA_HOME` environment variable. You can also specify a different default OpenAccess installation by using the *Configure* utility available though InstallScape. For information about both these options, see the *OpenAccess Configuration Guide*.

How Programmers Can Access Data



Examples of accessing the OpenAccess database.

Access using SKILL:

```
techGetSpacingRules(
 d_techFileID
)
```

Access using C on OpenAccess:

```
techGetSpacingRule(
   techFileId techfile,
   String ruleName,
   techLayerNum layer1Num,
   techPurpose purpose1,
   techLayerNum layer2Num,
   techPurpose purpose2,
   techPropType *pType,
   techPropValue *pValue,
);
```

Access using OpenAccess C++:

```
const oaString & oaLayerConstraintType::getName ( ) const [inline]
```

New, Changed, and Removed Functions

For information about new, changed, and obsolete C functions, see the *Cadence Integrator's Toolkit Database What's New* documentation.

For information about new, changed, and obsolete SKILL functions, see the *Virtuoso Design Environment SKILL Reference*.

Finding the Database, Build, and Release Number

Determining the Database Type

Use the C function `dbGetDatabaseType` to determine the database type. The function returns either `dbcCDBADbType` or `dbcOpenAccessDbType`, depending on the build hierarchy against which the code was compiled.

The SKILL function `dbGetDatabaseType` returns a string that is either `"CDBA"` or `"OpenAccess"` depending on the database being supported.

Because the `dbGetDatabaseType` SKILL function is relatively new for both CDB and OpenAccess, you can include a test for its availability in your database-specific code. For example, the following SKILL code conditionally handles a database-specific variation in database unit access:

```
if( ! getd('dbGetDatabaseType) || dbGetDatabaseType() == "CDBA" then
    bag = dbOpenBag( lib "a" )
    lib~>DBUPerUU~>maskLayout = 2000.0          ; CDB code
    dbSaveBag()
    dbCloseBag()
else            ; OpenAccess
    techSetDBUPerUU(techGetTechFile(lib) "maskLayout" 2000.0); OpenAccess code
)
```

This code works both on CDB and on OpenAccess.

Finding the Build and Release Number

Use the following functions to check the build information and release number of the current database:

- `dbGetBuildInfo`
- `dbGetVersion`

Finding the Database Type for a Virtuoso Executable

You can determine the database type that a Virtuoso executable will read or write by typing the `-dbType` option in a terminal window, as follows:

`name_of_executable` `-dbType`

This option returns either OPENACCESS or CDB.

For example, to find the database type for the `virtuoso` executable, type the following command in a terminal window:

```
virtuoso -dbType
```

Finding the OpenAccess Version

The *`installDir`*`/tools/bin/cdsGetOAVersion` script can be used to display the OpenAccess database component versions. The following is an example of the return value of `cdsGetOAVersion`.

```
Library:    oaBase      22.04-005    Tue Sep 11 00:00:17 2007
Library:    oaDM        22.04-005    Tue Sep 11 00:00:17 2007
Library:    oaDesign    22.04-005    Tue Sep 11 00:00:17 2007
Library:    oaTech      22.04-005    Tue Sep 11 00:00:17 2007
Library:    oaWafer     22.04-005    Tue Sep 11 00:00:17 2007
```

Finding the OpenAccess Data Model Version

You can run the standalone `cdsPrintOAFeatures` utility on a design or technology database to find its OpenAccess data model level and the associated features.

Syntax

```
cdsPrintOAFeatures -lib <t_libName> <options>
```

`-lib <t_libName>`	Specifies the name of the OpenAccess library whose data model level is to be queried. If cellview is specified, the command returns the data model level of the specified cellview. If no cellview is specified, the command returns the data model level of the technology database of the specified library.

`<options>` include,

`-cell <t_cellName>`	Specifies the name of the cell to be queried. Use wildcard character (*) to match all cells.
`-view <t_viewName>`	Specifies the name of the view to be queried. Use wildcard character (*) to match all views.

Note: Use a string quote or an escape character to prevent UNIX shell from expanding the wildcard character in your command arguments. For example, use `-cell "*"` or `-cell \*`.

`-level <n_dmLevel>[+]`	Specifies the level of the data model whose features are to be printed. Use the plus sign (+) optionally to print features of the specified and above levels. By default, features for all levels are printed.
`-V`	Returns the Cadence release version.
`-W`	Returns the Cadence release sub-version.
`-help`	Displays this list of options and their descriptions.

Examples

- To query a technology database:

  ```
  cdsPrintOAFeatures -lib gpdk090
  ```

  Returns:

```
Data model revision of tech gpdk090: DM 4
List of Features:
           DM 2: New Constraint
           DM 4: Purpose Aware Constraint
```

- To query a design database:

  ```
  cdsPrintOAFeatures -lib tiny -cell tiny -view layout
  ```

  Returns:

  ```
  Data model revision of design tiny tiny layout: DM 0
  List of Features:
  ```

  No list of features is returned implying that the cellview has no features.

- To query all layout cellviews at data model level 2 and above:

  ```
  cdsPrintOAFeatures -lib gpdk090 -cell \* -view layout -level 2+
  ```

  Returns:

  ```
  Data model revision of design gpdk090 cell_2 layout: DM 4
  List of Features:
             DM 4: New Value Type

  Data model revision of design gpdk090 cell_5 layout: DM 2
  List of Features:
             DM 2: Reference to FigGroup
  ```

Using Shareable OpenAccess Libraries

Shareable OpenAccess libraries are supplied in *your_install_dir*/tools/lib directory of the Cadence software. The .so extension applies to the Solaris operating system.

Translators

Converting CDB Data

Support for the cdb2oa translator was removed from IC6.1.8 ISR13. If you still require to migrate design data from CDB to OpenAccess, contact Cadence Customer Support to discuss how to proceed.

Virtuoso Studio Design Environment Specific Translators

The following translators are the Virtuoso on OpenAccess custom translators which provide the required technology file information and physical data for Virtuoso specific flows. These translators are the customized applications for the Virtuoso translators created by extending the base C++ classes, APIs, and the infrastructure supplied by OpenAccess libraries.

LEFDEF: `lefin`, `defin`, `lefout`, `defout`

XStream: `strmin`, `strmout`, `strminui`, `strmoutui`

The following translators are the OpenAccess translators that are released as part of OpenAccess tarkits and can be found in the *`your_inst_dir`*`/tools/bin` of the respective hierarchies. These translators accomplish only the basic Stream In/Out and LEF/DEF In/Out functionality and do not create Virtuoso specific data needed for display.

LEFDEF: `lef2oa`, `def2oa`, `oa2lef`, `oa2def`

XStream: `oaStrmIn`, `oaStrmOut`, `oaStrmInUI`, `oaStrmOutUI`

XStream Translator Replaces PIPO

XStream, a high-performance Stream for OpenAccess, replaces the Stream translator PIPO. XStream has been developed using the native OpenAccess APIs.

For information about how stream translation rules are mapped between CDB and OpenAccess, see Conversion of Technology File devices Class.

For information about the differences in the functionality and use model of PIPO and XStream, see Migrating from PIPO to XStream on page 389.

For information about the XStream and LEF/DEF translators, see the *Design Data Translator's Reference*.

Application

Virtuoso Chip Assembly Router

The Virtuoso Chip Assembly Router reads and writes OpenAccess 2.2 designs. The router operates primarily on a single, top-level cellview in the OpenAccess database. Routing rules for the Virtuoso Chip Assembly Router can be stored in the CDB technology file or in a separate ASCII file.

Encounter

The Encounter 4.2.1 software supports OpenAccess 2.2. For information about saving, importing, and restoring OpenAccess designs in the Encounter software, see *Encounter User Guide* version 4.2.1. Encounter 4.2 release requires OpenAccess version 2.2.4 or later.

2

OpenAccess Adoption with CDBA

This document contains information for Virtuoso Studio design environment application developers updating their code, tests, and data to work with OpenAccess.

Development Environment

The OpenAccess infrastructure packages are written in C++. Header files exported by the OpenAccess-based CDBA procedural interface have been changed to make them more ANSI-C compliant and compatible in C++ code. Linking executables require the C++ linker (CC).

Conditionally Compiling Code

The convention for conditionally compiling code differs on CDB and OpenAccess. To compile code, use an `ifdef` statement similar to the following:

```
#ifdef OPENACCESS
    OpenAccess_version_of_code
#else
    CDBA_version_of_code
#endif
```

To pass the `OPENACCESS` directive to the code, add `-DOPENACCESS` to the compile line. For example,

```
CC -DOPENACCESS myFile.cpp
```

Using OpenAccess Code and Extensions

The OpenAccess database is structured to be used in many different environments; each environment has its own requirements. Most of the time, you will be able to use native OpenAccess API functions to create, access, and manipulate database objects in your specific environment. For the few areas that are specific to an environment or graphical application, you will need application extension code to fulfill the unique requirements of the applications you are using.

For example, Virtuoso and third-party vendor applications might display data differently. These display methods are independent of the data itself. For the Virtuoso Studio design environment, there is a set of OpenAccess application extension objects containing information that controls the Virtuoso environment on OpenAccess, including layer-purpose pairs, display packets, selectability, and visibility.

For the Virtuoso Studio design environment, layer-purpose pairs (LPPs) are stored as application extension data. The public C-level emulation functions (for example, `techCreateLP` and `techAddLP`) handle internal conventions and extension data management; other C-level functions control layer-purpose attribute information, including drawing priority, packet, selectability, and visibility.

Examples of situations requiring application extension code:

- The OpenAccess extensions for technology LPP information such as packetName and priority are not included in native OpenAccess technology database definitions. This LPP information is required only by Virtuoso. Therefore, a technology file and database generated using native OpenAccess API functions, will not be able to view this LPP-specific information stored as OpenAccess extensions with the Virtuoso® editors, even though the data is there.

  To solve this problem, use Integrator's Toolkit (ITK) public functions (or SKILL) to add the required Virtuoso-specific additions/extensions for data such as LPPs. Then build on your Virtuoso-specific code to create the `display.drf` file required by Virtuoso.

  Note: C++-level functionality is not available for LPPs. You must use C functions through ITK.

  For information about ITK functions, see the *Cadence Integrator's Toolkit Database Reference*.

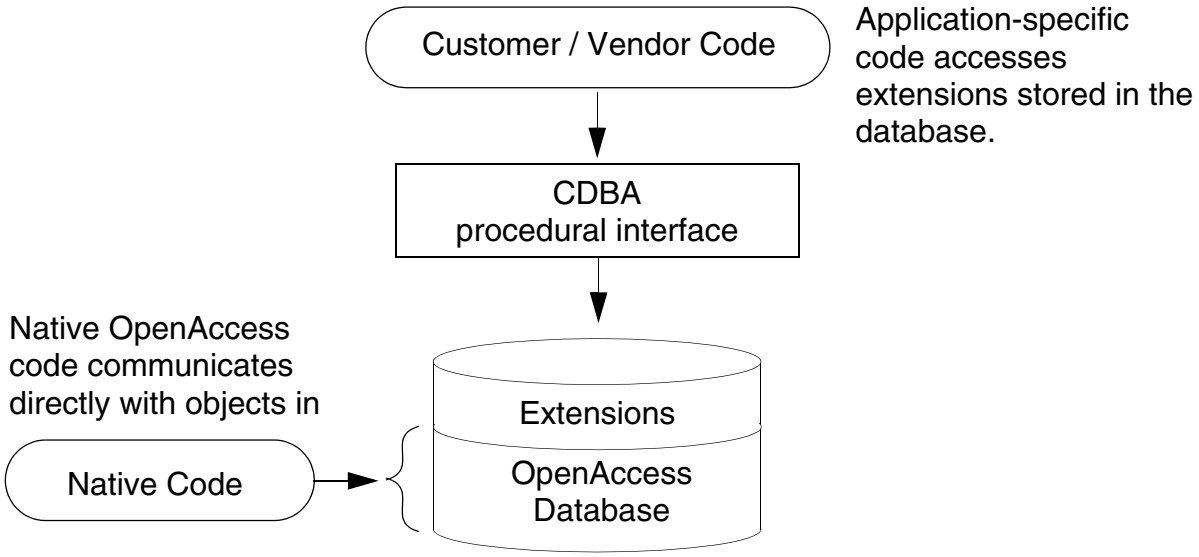
- Virtuoso Studio design environment LEF/DEF and Stream translators were written to share common functionality from core OpenAccess database requirements. New versions of the translators can be created and customized to handle the specific requirements of any vendor's application environment and extensions. The native OpenAccess LEF translator (`lef2oa`) and the customized version, `lefin`, both share

the same code base; the `lefin` translator was extended to handle Virtuoso-specific requirements, including layer-purpose and special via processing.

Structuring Your Code

Code written to support an application needs to be kept separate from code written using OpenAccess APIs. This allows code written to support an environment to be updated without impacting the OpenAccess code.

Native OpenAccess code interfaces directly with the objects in the OpenAccess database. Application-specific code can access the extensions that are also stored in the OpenAccess database. The Virtuoso Studio design environment uses application-specific extensions.



Application-specific information can also be stored in separate files, such as the Virtuoso-specific `display.drf` ASCII file.

3

CDB/OpenAccess Database Differences

This chapter describes differences in behavior between CDB and OpenAccess for the following subjects:

General Differences

Using Function Calls on Standard Vias

In the Virtuoso Studio design environment, the masters of standard vias are not associated with a library, cell, or view name. A viaSubMaster is opened by OpenAccess in virtual memory only.

The SKILL API `dbIsCellViewStdViaMaster` is used to determine whether a cellview is a standard via master.

The following return values are given for SKILL and C functions calls to standard vias.

Using Database (db) Function Calls

Function	Return Value
`dbGetCellViewLibId`	Returns NULL ID
`dbGetCellViewDdId`	Returns NULL ID
`dbReOpen`	Returns NULL and gives a warning
`dbCellViewNeedRefresh`	Returns FALSE
`dbGetCellViewFullDirPath`	Returns NULL
`dbClose`	Only closes viaSubMaster
`dbGetCellViewLibName`	Returns the techLibName where the standard via definition is defined
`dbGetCellViewCellName`	Returns the standard via definition name
`dbGetCellViewViewName`	Returns the view name

Using Design Management (dd) Function Calls

A NULL value is returned when attempting to use `dd` function calls to return the master ID of standard vias.

Calls to dbRefreshCellView

The function `dbRefreshCellView` has been changed on OpenAccess such that, when a cellview in virtual memory is updated with the copy from disk, the access mode is changed to `'a'`. In CDB, the access mode was not changed by `dbRefreshCellView`.

This change has been incorporated because the method in which the OpenAccess database refreshes a design is mode dependent. The design is returned to the state it was in when it was first opened. If the design was opened in '`w`' mode, it will be truncated with no reloading from disk.

Using the Schematic Bus Range Order Option

In CDB, changing the Virtuoso Schematic Editor option, *Bus Range Order* in the *Editor Options* form, would prompt you to save your edits. This was also seen when using the SKILL function `dbSetNetNameDescend` which sets the direction of the net name vector expression

to descending or ascending. In CDB, changing the bus range order would indicate that the cellview had been modified and therefore needed to be saved.

In the Virtuoso Studio design environment, when changing the bus range order using the *Bus Range Order* option or `dbSetNetNameDescend`, you will not be prompted to save changes. The cellview read-in counter does not change, which means the cellview is not flagged as modified.

Calls to dbReplaceProp for Abutment Groups

When using the function `dbReplaceProp`, you must call `dbReplaceProp` for abutment groups before calls to `dbReplaceProp` for Pcell properties. This is true for CDB and the Virtuoso Studio design environment. The ID of the objects before and after abutment occurs will not be the same. In some situations on CDB, it was possible to reference the same pin IDs and instance masters in virtual memory after Pcell re-evaluation, but because of changes to complex pin models and when data is purged in OpenAccess (see Complex Pin Modeling on page 49 and Instance Masters on OpenAccess on page 44) you can not rely on the same data or IDs after Pcell re-evaluation.

Mark Net Command

In CDB, the *Mark Net* command relied on the `viaLayers` section of the technology file to determine interconnecting layers.

In the Virtuoso Studio design environment, the *Mark Net* command relies on the `viaDefs` section of the technology file to determine interconnecting layers. Most CDB technology file contacts device definitions are converted to via master definitions (`viaDefs`) in OpenAccess. Via definitions consist of two types of vias, `customViaDefs` and `standardViaDefs`. For more information about via definitions, see Technology File Via Definitions and Via Specifications on page 343.

Enhancements in the Design Summary Form

The *Design Summary* form has been enhanced to show, row objects, wire statistics, marker objects, track patterns, blockage objects and boundary objects.

Note: The *Instance Statistics* section of the *Design – Summary* form displays the number of unplaced instances. The `unplaced` attribute is maintained by P&R applications. An unplaced instance is not displayed in the Virtuoso layout window, it is however present in the database. See Placement Status on page 69 for more information.

Calls to the cdsInit and dbInit Functions

Programs using the OpenAccess C++ layer must call the `cdsInit` or `cdsInitNonCDSApp` function explicitly in order to specify how the lower-level library search mechanism will work. When using ITKdb, the `dbInit` function must be explicitly called. The `dbInit` function registers the SKILL language interface so that Pcells will work.

Defragmentation in OpenAccess

The `dbDefragCellView` function has not been implemented in Virtuoso Studio design environment. The user interface command and SKILL function are not available.

Maximum Cellview Bounding Box

OpenAccess and the Virtuoso Studio design environment allow a larger range of coordinates for cellviews. This limit is stored in the constant `dbcMaxCoord`.

- OpenAccess and the Virtuoso Studio design environment let you create 32-bit coordinates in the range of

  -(2^{31} - 1) through +(2^{31} -1) database units

  (or in hexadecimal, `7fff ffff`).

- In OpenAccess is:

  -2,147,483.65 through +2,147,483.65 user units

The Virtuoso Studio design environment currently limits the windowing range to approximately plus or minus 700,000 user units.

Name Space Differences

The OpenAccess name space is different from the CDB name space. For example, in OpenAccess, square brackets are used as the vector expression delimiters for bus names. The CDBA procedural interface will continue to present the CDB type name space, including the use of “< >” for vector indexes.

An `oaName` object encapsulates all of the mechanics of name mapping. This effectively hides the internal representation of the name from the caller and allows mapping between name spaces.

Database Size Differences Using Paths Versus PathSegs

Converting CDB data containing paths to OpenAccess data with paths creates an OpenAccess database larger in size than the CDB database. The paths explode the size of the database, especially in the case of digital data. In contrast, OpenAccess data with pathSegs creates a size-efficient OpenAccess database. Therefore, when converting data from CDB, DEF or GDSII to OpenAccess, always use the options to create pathSegs instead of two-point orthogonal paths, especially for digital data.

In addition, it is recommended to bring in via placements after conversion as vias instead of cell placements. This may require you to create an ITDB library for the design, which can hold the required standard and custom via definitions. This will also shrink the OpenAccess database to smaller than the CDB data.

Escape Sequences in Names

Except for pin names, escape sequences are now required in names to represent a printable character.

For example,

```
a\142c
```

is a legal name because it is the same as “abc”, but

```
a\006c
```

is not a legal name, because the `\006` represents the `ACK` control character, which is not printable.

Backquote Character in Names

Tools that convert data from a logical design to physical data frequently need to flatten the logical hierarchy. In doing so, they attempt to create names for objects that make it easier to map the objects back to their original logical hierarchy. One example of this is PRflatten. This tool takes the original path names to objects, such as

```
"/top_inst/mid_inst/netA"
```

and convert them to names such as

```
"|top_inst|mid_inst|netA"
```

in a single flat cellview. The pipe character (|) in this flat name is used to represent the original hierarchy and is therefore called a *pseudohierarchy* character.

Note: Limitations on the use of special characters does not apply to pin names; a pin name is a string for which there is no syntax check.

OpenAccess formally supports the concept of pseudo hierarchy in name mapping. The CDBA procedural interface uses the backquote character (` ) as the pseudo hierarchy character. Therefore, do not use the backquote character ( `) in names unless you intend that it be interpreted as pseudo hierarchy by other tools, such as the DEF translator. The pipe character (|) is used to represent pseudo hierarchy by convention between the PRflatten and Preview tools.

On many keyboards, the backquote character (`) is located under the tilde character (~). In ASCII code, the backquote is represented by the following numbers:

hexadecimal	60
octal	140
decimal	96

Translators in the physical domain will consider the backquote character (`) as hierarchy when mapping to languages, such as spef and SDF, that treat pseudo hierarchy as real hierarchy.

In the ASCII technology file, the single quote (forward tick ') is the SKILL sign for symbol. Do not use the backquote. For example,

```
orderedSpacings(
("minOppExtension" "poly1" "metal2" (2.1 0.7) 'soft)
...
)
```

Using Instance Parameters

If an instance of a Pcell has instance parameters and CDF parameters that overlap in their names, types, and default values, you can use the *Store Default* attribute to determine which of the two parameters should take precedence.

If you want the CDF parameter of a Pcell to use the evaluated result of an expression string, set its *Parse as CEL* value to *yes*. Also set the `CDS_NETLISTING_MODE` value to `Analog`.

Example

Consider the following NLP expression as a CDF parameter of a Pcell instance:

```
[@nw]
```

where, nw has been set as `nw = "3u"`.

In this situation if *Parse as CEL* value is *yes*, the instance parameter will be set to `"3u"`. (If *Parse as number* is also set to *yes*, the expression will be evaluated to `3e-06`). If *Parse as CEL* is set to *no*, then the parameter expression will not be evaluated and will be set to `"[@nw]"`.

You can update both *Store Default* and *Parse as CEL* values in the *Edit CDF* form (In the CIW, click *Tools - CDF - Edit*).

Instance Magnification Differences

OpenAccess does not support instance magnification.

The *Property Editor* and *Create Instance* forms no longer contain fields for magnification. Both the Virtuoso Layout L Editor and the Virtuoso Schematic L Editor handle magnification on OpenAccess as described below.

The Virtuoso Studio design environment supports the translation of magnified instances, including that of hierarchical and Pcell instances, from CDB and Stream to OpenAccess. Translation is done by creating special parameterized cells that emulate magnification and any-angle rotation. In the future, this method will facilitate better handling of all angles of rotated instances (srefs) in Stream.

Each of these special Pcells contain SKILL code that:

- Creates an instance of the original instance master.
- Flattens the instance.
- Transforms all the resultant shapes by the magnification of the original instance (the `mag` parameter of this Pcell).

The Pcell master created in OpenAccess for a translated magnified instance will have library, cell, and view names based on the names of the original instance master. The cell name will be identical to the original, and the view name will be appended with `_xform`.

If the library of the original instance's master is writable, it is used; otherwise, the library of the cellview in which the instance is instantiated is used.

For example, when the master of the magnified instance is

(`A_lib B-cell symbol`)

then the Pcell master is

(`A_lib`, `B_cell`, `symbol_xform`)

The Pcell master has two parameters, *`mag`* and *`angle`*.

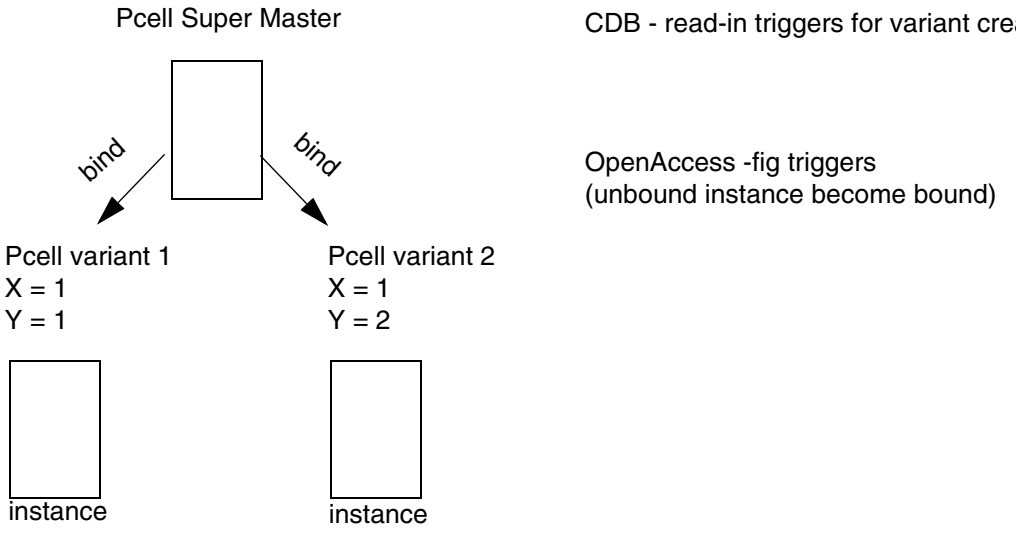
When you run a netlister, the current method of translation of magnified instances involves two restrictions:

- The *`viewname_xform`* cellview should be translated to the instance's master library. This means you MUST have write permission for the instance's master library.
- You have to add *`viewName`*`_xform` to the netlister's view name list before running the netlister, for example, add `symbol_xform` to the view name list.

ITK Read-in Triggers

Customer application code that relies on read-in triggers to fire for instance creation, must now use fig triggers for instance binding. On OpenAccess, fig triggers are fired in the event of instance binding changes and are not fired for Pcell submasters evaluation.

When triggers are fired	OpenAccess	CDB
read-in triggers fired (variant creation)	No	Yes
fig triggers fired (instance binding)	Yes	No



Instance Masters on OpenAccess

CDB did not purge instance masters even when there were no instances referring to the master. In the Virtuoso Studio design environment, the instance master is purged as soon as the instance reference count associated with the instance master becomes zero.

On CDB, an instHeader ID remains the same when making changes to the instHeader master. On OpenAccess, there is no equivalent of setting the master for instHeader natively. In the Virtuoso Studio design environment, this is emulated by setting the master of individual instance on the instHeader to the new master. All the instances are removed from the old instHeader and moved to the new instHeader, which could or could not exist.

However, when an instHeader is empty, meaning it does not have any instance on this instHeader, the instHeader is removed and the ID is no longer valid. This change has effected the behavior of the SKILL function `dbSetInstHeaderMaster`.

The following code is an example of retrieving the instHeader after setting the master. In this example, the instance should be retrieved first. Then, set the instHeader after the master is reset.

```
inst = car(ih~>instances);
dbSetInstHeaderMasterName(ih newLib ih~>cellName ih~>viewName);
set ih = inst~>instHeader; ;; Changes to instHeader
```

Order of Generating Objects

In CDB, the order in which objects were returned by the different generators depended on the type of object and whether the objects was new and had not yet been saved to disk. If a number of objects were created, they were generated in the order they were created. However, after the data had been saved, purged from memory, and then opened and read in, the order would be different than before but consistent from this point forward.

OpenAccess is consistent in the order that objects are generated. The generators always generate shapes in the same order.

Undo/Redo

The following table describes the differences for the *Undo* and *Redo* commands on CDB and the Virtuoso Studio design environment.

Behavior	Virtuoso Studio design environment	CDB Database
Supports the undo/redo of changes to parameterized cell parameters on instances, including stretchable Pcells?	yes	no
Number of undo levels	0 to 128 levels	0 to 10 levels
Prompts you to save?	Only if the cellview has been changed	Yes, if the cellview has been edited, even if undo commands have restored the cellview to its original state.
Behavior when all the changes to a cellview have been undone.	`dbIsCellViewModified` returns FALSE and the cellview attribute `modifiedButNotSaved` returns `nil`	`dbIsCellViewModified` returns TRUE and the cellview attribute `modifiedButNotSaved` returns `t`

techFindViaDefByName vs leIsContactName SKILL API

The `techFindViaDefByName` API is available in OpenAccess (OA) and its corresponding API is `leIsContactName`, which was only supported in CDB.

Some observations on inconsistency in CDB and OA translation are given below regarding via and contacts across LEF, VLE, and technology file.

1. If via, `via1` is displayed as `VIA DEFAULT` or `VIA` in the LEF file, then, after LEF In:

CDB	OA
It gets mapped to `symContactDevice` in technology file.	It gets mapped to `customViaDef` and `LEFDefaultRouteSpec` in technology file.
Shows up as a contact in the list in the Create Contact form in VLE.	Does not show up in the list in the Create Contact form in VLE.

2. If via, `via2` is displayed as `VIARULE GENERATE` in the LEF file, then, after LEF In:

CDB	OA
It gets mapped to `prGenViaRules` in technology file.	It gets mapped to `stdViaDef` and `cdsVia`.
Does not show up in the list in the Create Contact form in VLE.	Shows up as a contact in the list in the Create Contact form in VLE.

Files

Set-up Files New in the Virtuoso Studio design environment

- `data.dm`

  The data.dm file is a design management data file used, in part, for storing properties when using the DMFileSys plug-in. For more information, see Properties of Libraries, Cells, and Views on page 79.

- `.oalib`

  The `.oalib` file is created to store information that is used by OpenAccess for library management. If a `.oalib` file does not exist when a library is opened by OpenAccess, it will be created. You must have write access to the library in order for a `.oalib` file to be created.

Files that Have Changed

- `layout.cdb`, `sch.cdb`, etc.

  Name change only. The file extension for design database files is `.oa`.

  In CDB, the cellview data filename did not need to reflect the view type stored in the cellview data file. For example, CDB allowed a cellview with the stored view type `maskLayout` to be called `symbol.cdb`.

  The Virtuoso Studio design environment expects a cellview filename to reflect the stored view type and will not allow a conflict in naming and type.

  For information about cellview type mapping, see Mapping of Cellview Types from CDB to OpenAccess on page 77.

- `tech.db`

  The technology database filename changes from `techfile.cds` to `tech.db`. Application code must be changed to no longer refer to the technology database file by

name. For information on how to retrieve the technology database filename, see Chapter 4, "OpenAccess Technology Data Changes."

- `pc.db`

  The `pc.db` file typically resides in the cellview directory and stores the hierarchical relationships for that cellview. It is a derived file that allows hierarchy functions to navigate descriptions other than CDB; for example, VHDL and Verilog.

Files that No Longer Exist

- `prop.xx`

  See Properties of Libraries, Cells, and Views on page 79.

- `lib.defs`

  The `lib.defs` file is no longer a supported library definition file. See Library Definition Files on page 71 for more information.

Files that Are the Same

- `cdsinfo.tag`
- *`name`*`.Cat` (category registry)
- `libInit.il` (SKILL loader)
- `master.tag`

  The `master.tag` file contains the name of the master (or primary) data file, and optionally, a comment such as the version number.

  A cellview will fail to open if it is created without a `master.tag` file. If you create a new cellview using the UNIX `mkdir` command and then copy the files into the cellview in a random order, you will not be copying the `master.tag` and master files first. If during the copy, you or some other process request OpenAccess to evaluate the cellview (and it does not contain a `master.tag` file) an error message will be issued.

Pins

Complex Pin Modeling

Converting CDB Subnet Pin Model to OpenAccess Pin Group Model on page 50

Attributes on Pins on page 51

Pin Group Conversion During Pcell Evaluation on page 52

How Undefined Pin Groups are Converted on page 53

Reloading Pcells After CDB to Virtuoso Studio design environment Translation on page 53

Changes to C Functions for OpenAccess Pin Group Model on page 54

In CDB, subnets are used to implement the pin group relations of strong, weak, or mustJoin. Subnet extensions are represented by a two level subnet structure. Pins are placed at the second level subnets and the connection status of the pin group is determined by their relative location in the structure. Also, there is a one-to-one relationship between the pin and the physical implementation figure.

Pin models in the Virtuoso Studio design environment allow multiple figures per pin, as well as multiple pins per terminal. Subnets are no longer used to represent pin group relationships.

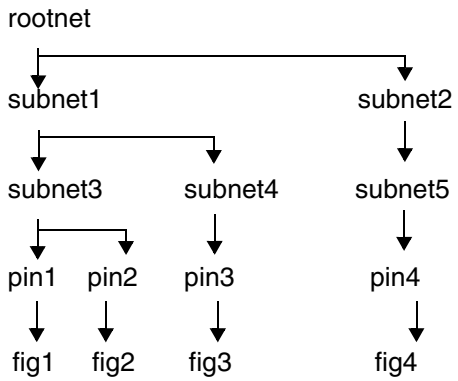
The `oaPinFig` class allows the figures (instances, shapes and vias) to be added to a pin. Pins can be created with 0 to *n* figures attached. When writing SKILL code, pins can be initially create with no figures. You can then add or remove figures from the pins in subsequent calls.

Important

Do not leave pins without a figure attached. Creating a pin without a figure is intended to be a transitional state, not a permanent, or saved state.

The following diagram describes CDB pin grouping versus the Virtuoso Studio design environment pin modeling.

CDB Subnets Model

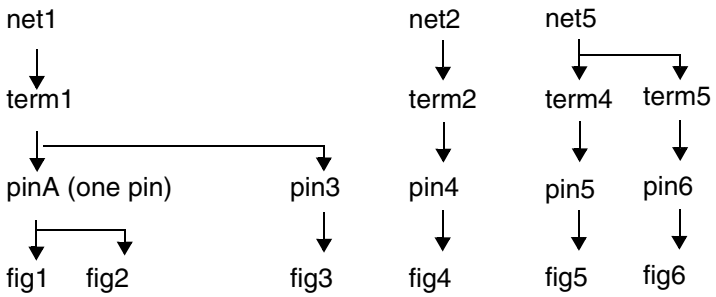


Figures on the same second level subnets are strongly connected: fig1 and fig2

Figures on different second level subnets and on the same first level subnet are weakly connected: {fig1, fig2, and fig3}

Figures on second level subnets and are on different first level subnets are must connect: fig3 and fig4

Virtuoso Studio design environment Model



Figures on the same pin are strongly connected: fig1 and fig2

Figures on different pins but on the same terminal are weakly connected: {fig1, fig2, and fig3}

Figures on terminals with different nets, and the terminals have the mustJoin attribute are must connect: fig3 and fig4

Figures on different terminals on the same net are shorted: fig5 and fig6

Note: You can indicate a must connect by setting the `mustJoin` attribute on the terminals.

Converting CDB Subnet Pin Model to OpenAccess Pin Group Model

Two types of situation exist where the CDB subnet pin group relationship is converted to the Virtuoso pin model.

- CDB data

  Subnet pin modeling is converted to OpenAccess pin modeling.

- SKILL Pcell code

  Any Pcell code dependent on the subnet structure is still valid in the Virtuoso environment. Once the Pcell code is evaluated, the subnet pin model in the subMaster is dynamically converted to OpenAccess pin model in virtual memory. This may slow

down the performance. Therefore, although it is not required, re-writing your Pcell code using the following functions to define and retrieve the name, access the direction, and the placement status of pins will provide better performance.

`dbGetPinFigName`

`dbGetPinFigAccessDirection`

`dbGetPinFigPlacementStatus`

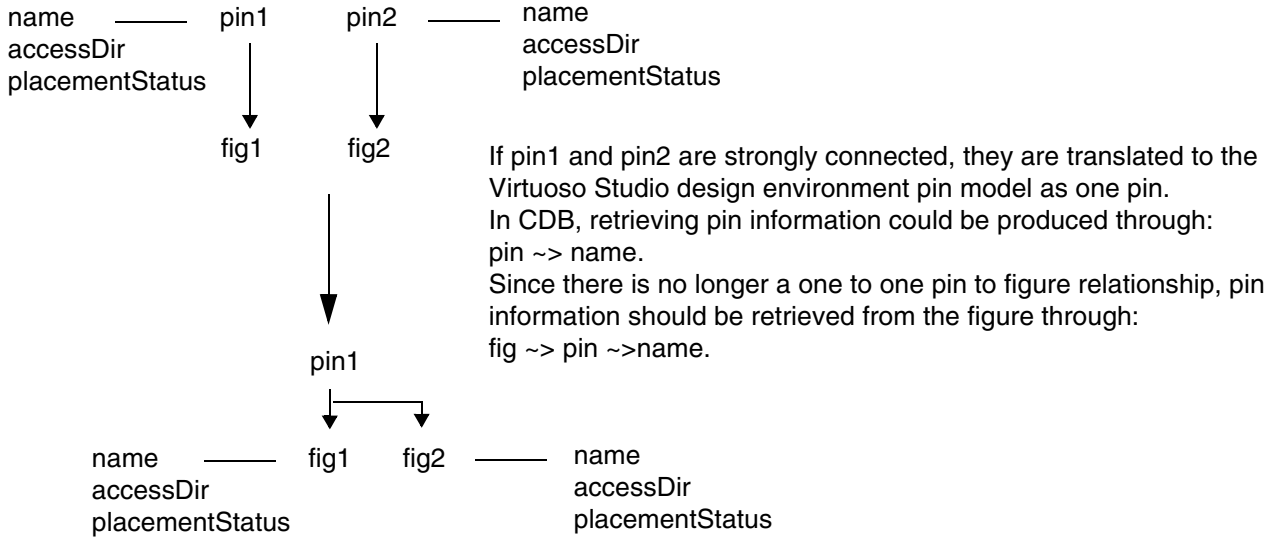
`dbSetPinFigName`

`dbSetPinFigAccessDirection`

`dbSetPinFigPlacementStatus`

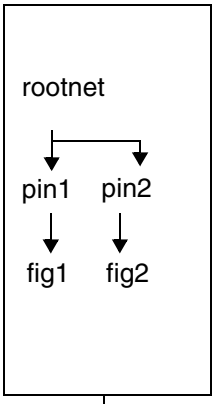
Attributes on Pins

In CDB, pin physical attributes are stored on the pin object even though they apply to the pin figure because of the one-to-one relationship between a pin and the physical implementation figure. These attributes are used to access the name, access direction, and placement status. During translation to the Virtuoso Studio design environment, the attribute information is moved down to the figure. The following functions are provided to access pin information from the figure: `dbGetPinFigName`, `dbGetPinFigAccessDirection`, `dbGetPinFigPlacementStatus`, `dbSetPinFigName`, `dbSetPinFigAccessDirection`, `dbSetPinFigPlacementStatus`.



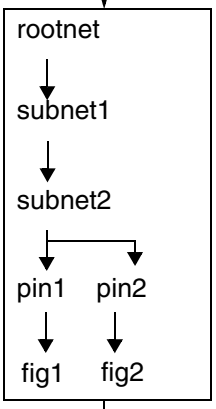
Pin Group Conversion During Pcell Evaluation

The following describes how CDB pin group connection status in the subMasters of Pcells are converted to the Virtuoso environment pin model.



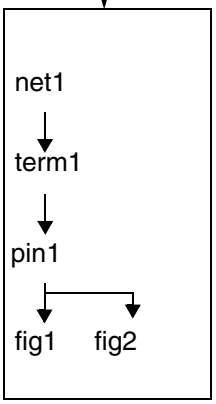
Pcell CDB Pin Model During Pcell Evaluation - Creating Pins

In CDB, a collection of figures can be created, and from the figures create a collection of pins. Based on the subnets, the pins and figures are given a connection status. There is one pin per figure.



Pcell CDB Pin Model During Pcell Evaluation - Establish Connection Groups

The Virtuoso Studio design environment creates or preserves subnet structure during Pcell evaluation, when the SKILL API dbStronglyConnectedPins, dbMustConnectedPins, dbExternallyConnectPins, or dbWeaklyConnectPins are used to grouping pins using subnets. During evaluation, pin ID are still valid.



Pcell Virtuoso Environment Pin Model - After Pcell Evaluation

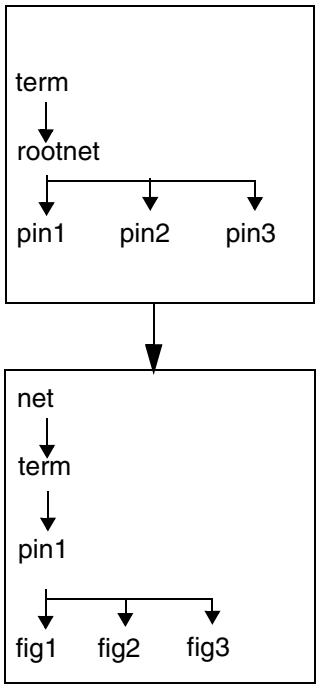
After Pcell evaluation, all subnets found are convert to the OpenAccess pin model. Original pin attributes are preserved on the pin figures. When a collection of pins (or their figures) are strongly connected (dbStronglyConnectedPins) all but one pin is deleted.

Important: The remaining pin could be replaced with a new pin, resulting in a new pin ID. If a new pin is created, code which attempts to reference the original pin IDs returns NULL. Pin information can be retrieved from the figure through:
fig ~> pin ~>name.

How Undefined Pin Groups are Converted

When Pcells are created in CDB without an explicit pin model (without subnets), by convention, the CDB application would interpret the connection status. In the case of VXL in CDB, these pins would be considered strongly connected.

To resolve ambiguous or undefined pin status, the property `defaultPinModel` with a value of strong is added to all super-masters of `maskLayout` cellviews which do not have explicit pin model information (subnets). This allows undefined DFII on CDB pins to be explicitly modeled as strongly connected in the Virtuoso Studio design environment. If the subnets do exist, the pin connection status is evaluated based on subnets.



Pcell CDB Pin Grouping - Undefined Pins in CDB

In CDB, if the subnet model does not exist, applications interprets the pins to be strongly connected.
In the Virtuoso Studio design environment, this same pin relationship is, by definition considered weakly connected.

Virtuoso Studio design environment Pin Model

In OpenAccess, pins on the same terminal are weak connect by definition.
If the defaultPinModel property (value of strong) is added to the supermaster of the Pcell, the figures of pins will be converted to strongly connected.

Reloading Pcells After CDB to Virtuoso Studio design environment Translation

When loading SKILL Pcells that do not explicitly state the connection status, you must either change your SKILL Pcells to explicitly describe the pin connection status or add the `defaultPinModel` property to the Pcell master. The recommended solution is to modify Pcells to correctly describe the pin model.

- To describe the pin connection status in SKILL Pcells, create pin figures using `dbCreatePin` and then add figures to the pin using `dbAddFigsToPin` (or `dbAddFigToPin` for adding a single figure). Define the pin figure properties by using `dbSetPinFigName`. Additionally, you can use `dbSetPinFigAccessDirection` and

`dbSetPinFigPlacementStatus` for defining complex connectivity model for Pcell abutment.

- If you want to temporary add the property to the Pcell master, define the `defaultPinModel` property in the technology file. For example,

```
tfcDefineDeviceProp(
(layout    m1_m2    defaultPinModel    "strong")
)
```

  Valid values for `defaultPinModel` are `strong`, `weak`, and `must`.

Changes to C Functions for OpenAccess Pin Group Model

`dbCreatePinOnTerm`

This function creates a new pin using a supplied name. In CDB, there was a one to one relationship between pin and figure. Pins could not be created without a figure and were uniquely named per the terminal.

In OpenAccess, it is possible to have a pin with 0 or more figures. When using `dbCreatePinOnTerm` the figure on a pin is now optional. If the figure is not supplied when creating a pin, an empty pin created. The behavior of `dbCreatePinOnTerm` is:

- Creates an empty pin if the named pin does not already exist.
- Creates an empty pin with a specified figure ID if the named pin does not already exist.
- Gives a warning and returns `dbcNullId` if the named pin already exist.

Note: To add a figure to a pin, use `dbAddFigToPin`.

For more information, see Pin Modeling SKILL functions and Pin Modeling C functions.

Symbolic Device Masters

Symbolic device masters (`syContact`, `syPin`, and `syRectPin`) are obsolete in the Virtuoso Studio design environment. When translating DFII on CDB technology data to the Virtuoso environment, device definitions are deleted from the technology file, device masters are optionally deleted from disk (Keeping Device Masters on page 341) and all instances are mapped to vias or converted to rectangular shapes with pins.

For information about how CDB symbolic devices masters are mapped and how to enable a preservation of symbolic device masters, see Mapping CDB Symbolic Device Masters on page 340.

Instance Pin (instPin) Objects

Because pins are now allowed to have several figures, the instance pin (instPin) object is obsolete. In CDB, an instPin was a database object that represents a pin of a terminal in the master of an instance. Any existing user code that refers to instance pins must be updated.

Note: This change does not imply that instances cannot be pins. Pins can be created as instances and you can specify an instance as a pin figure. In CDB, the master of instances that were pins were typically `syPin` devices. `syPin` devices are now converted to rectangles as pins in the Virtuoso Studio design environment. See Symbolic Device Masters on page 54.

InstTerms and Terminals

Iterated Instances

If an instance is changed to be an iterated instance, for example `I1` to `I1<0>`, the object type is changed. Be aware that object types such as `I1` are a scalar instance and if changed to `I1<0>`, the object type is a vector instance. User SKILL or C code that change instances to be an iterated instances, will no longer return the same object ID.

Note: The angle brackets <> are use for iteration of instances as well as bits of a nets.

Net Width and Terminal Width

When an instance terminal (instTerm) is created, the net width and terminal width must be the same. If they are different, a warning is issued indicating that the net width does not match the instance/terminal width.

Terminal Required on Instances

To allow an instance to be assigned to a net in OpenAccess, an instance terminal is created on the instance and connects that instance terminal to the net. However, this can only be done if the instance master has one or more terminals. If you need to be able to associate an instance with nets, add at least one terminal to the instance master in CDB and then run the translator again.

Note: This does not apply to vias, as vias are no longer instances.

Terminal Ordering

In the Virtuoso Studio design environment, terminals can be created and referenced by position rather than by name only. This allows better support to the connection semantics of Verilog and other netlist formats.

In the following example, the terminals are referenced by their position for connectivity, or the order in which they are specified in the Verilog statement.

```
subModule M1(net1, net2)
```

`M1` is the instance of the master `subModule`. `net1` is connected to the first terminal, and `net2` is connected to the second terminal. Even if the names of the terminals are change in the master, the connection is reference by the terminal order.

To further support terminal ordering, the following functionality is supported at the API level.

- Assign the position to a terminal
- Remove the position from a terminal
- Query the position of a terminal
- Find a terminal based on position
- Create an instTerm based on position
- Find an instTerm based on position
- Query an instTerm terminal based on position

Note: All instTerms associated with an instance must be connected by reference to the name or to the position.

For more information, see Terminal Ordering SKILL Functions and Terminal Ordering C Functions.

Bus and Bundle Nets

Bus Nets, Bundle Net, and Bus Net Bit

CDB connectivity is represented by signals, the nets that implement those signals, instance terminals, and terminals. In OpenAccess, there is a net base class from which are derived the scalar and multi-bit nets.

- The bus net is a multi-bit net that represents a collection of logical connections. It is associated with a single base name and vector range specification. A bus net has a corresponding `busNetDef` that manages all bus nets that share the same base name.
- The bundle net is a multi-bit net composed of one or more scalar nets, bus nets, or a combination of both. The bundle net object can be viewed as a collection of single bit net members and represents a collection of logical connections. The index and ordering of each member of a bundle net is indicated by the bundle name representation.
- The bus net bit represents a single bit of an bus net. When a bus net is created, a corresponding set of bus net bits are automatically created. An implicitly created bus net bit cannot be used to connect to terminals or instTerms, nor can it be associated with shapes or routes.

You can change a net from implicit to explicit by creating a net with the same name as an implicit net or by scalarizing the multi-bit nets. If you scalarize a multi-bit net, all the associated implicit bitNets become explicit and the multi-bit net becomes implicit. A scalarized multi-bit net can be referenced, but not modified.

Bus Definitions

There are three types of bus definitions objects which define the vectored objects busNet, `busTerm`, and `vectorInst`. The definition includes base name, bit range, and bit order. The same bus definition manages all vectored objects in a block with the same base name.

The bus definition class tracks the minimum and maximum indexes referred to by all the corresponding bus and busBit objects. A bus definition can have missing bits. Buses need not start or end at zero, but the indexes must be greater than or equal to zero.

If an implicit or explicit bus definition object does not already exist when a bus object is created, the database automatically creates an implicit bus definition object. The database adds vectored objects with the same base name to the vector definition.

For more information, see Bus Definition SKILL Functions and Bus Definition C Functions.

Nets with Equivalent Names

The overall connectivity is the same between OpenAccess and CDB. However, the two databases store net names differently, and OpenAccess does not support complex bundle names.

How Net Name Variations Are Stored

The variations on a net name are stored as follows:

- CDB stores each variation on a net name separately
- OpenAccess stores all equivalent variations on a net name as one net name

For example, when you create the following three nets:

a
<*1>a
(a)

which all have `a` in common, CDB stores each of the three variations separately, while OpenAccess stores all three as one net name, `a`.

As a second example, the following net names:

```
a,b
(a),b
a,(b)
(a),(b)
```

are stored as four different names by CDB, but by OpenAccess a single net name:

`a, b`

Complex Bundle Names Not Supported by OpenAccess

OpenAccess does not support complex bundle names, including repeated or nested bundle names. OpenAccess expands complex bundle names. If a bundle net or terminal is a complex name, OpenAccess returns the expanded name the next time it is queried. For example, OpenAccess expands the following repeated bundle name,

```
<*1024>(a,b)
```

to 1024 occurrences of a, b, such as this:

```
a,b,a,b,a,b,a,b,a,b,...
```

and OpenAccess expands the following nested bundle name,

```
gnd!,<*1024>(a,b)
```

to gnd!, followed by 1024 occurrences of a, b, such as this:

```
gnd!,a,b,a,b,a,b,a,b,a,b,...
```

Looking for a Net by Name

When you look for a net by name using the `dbFindNetByName` function, the results are different in CDB and the Virtuoso Studio design environment. In CDB, `dbFindNetByName` returns a different result for each variation of the net name (or fails if the name is not a literal match), while on OpenAccess, `dbFindNetByName` returns a single net for all variations of the name that are equivalent.

Note: `dbFindNetByName` can not be used to find the net name of subnets. `dbFindNetByName` only returns the matching root net name.

For example, when `dbFindNetByName` on CDB returns

```
<*1>a
```

`dbFindNetByName` on OpenAccess returns

```
a
```

SKILL Access for Nets/Signals

CDB distinguishes between nets and signals. In CDB, querying attributes with SKILL returns different attributes for nets than for signals. In OpenAccess, there are no signals. The various signal-specified functions in the CDBA procedural interface will process all single-bit nets as a signal. The attribute access has been modified in the following way:

- For a multi-bit net, only net attributes are presented/processed.
- For implicit single-bit nets (for example, scalar members of a `bundleNet` that do not exist as independent scalar nets), only signal attributes are presented/processed.
- For single-bit nets, both net and signal attributes will be presented/processed.

Net Status Attribute

The *Net Status* attribute has been removed from the Property Editor form for OpenAccess because OpenAccess does not support this attribute.

Embedded Module Hierarchy (EMH)

Embedded Module Hierarchy (EMH), is a model for storing hierarchical design data supported at the OpenAccess database level.

EMH lets applications preserve a consistent relationship between the logical and physical hierarchy through the implementation flow, and it provides an occurrence model for annotating data specific to each distinct unfolded object in a design.

For information about specific OpenAccess support, go to the following URL. Login accounts are required for file downloads and many site features.
http://openeda.si2.org

Inherited Connections

In CDB, an inherited connection is created when a *net expression* is associated with either a signal or a terminal. The connection is overridden by a `netSet` property placed on an instance somewhere in the hierarchy above. The net expression specifies the name of a `netSet` property to search for in the hierarchy above and a default name to use.

- In the case of an inherited signal, the net expression specifies the name to use for the corresponding signal.
- In the case of an inherited terminal, the net expression specifies the name of the net that is to connect externally to the terminal if there is no corresponding physical connection through an instance terminal.

CDBA emulates a net that has an associated `oaNetConnectDef` as an inherited signal and a terminal with an associated `oaTermConnectDef` as an inherited terminal.

Mosaics

CDB supports objects called mosaics, which are two-dimensional arrays of cellview placements. They can be simple or complex. A simple mosaic has only one type of instance master cellview, and the space between its rows and between its columns is the same. A complex mosaic can have a variety of instance master cellviews at different locations in the array, and the space between its rows and between its columns can vary.

Complex mosaics were created by the structure compiler application. For a description of simple and complex mosaics, see the *Cadence Integrator's Toolkit Database Reference*.

OpenAccess does not support mosaics in the same way that CDB does. OpenAccess has only instances, which can have row and column attributes of one or greater. When the attribute of a row or column is greater than one, the instance is an called an *arrayed instance*. CDB simple mosaics are emulated by using OpenAccess arrayed instances.

Naming of Mosaic Instances

For mosaics, CDB uses more instance names than OpenAccess uses. CDB assigns mosaics a unique name of the format `M`*`n`*, where *`n`* is 1 for the first mosaic and *`n`*+1 for the subsequent mosaics; CDB then assigns the resulting mosaic instance the name `I`*`n`*+1. OpenAccess assigns mosaics a unique name of the format `M`*`n`*, and assigns the same name to the resulting mosaic instance.

For example, if you open a cellview in the layout editor and use the Create Instance command to create two instances, then a mosaic, and then another instance, the resulting default instance names are as follows:

- For CDB:

  First instance is named `I0`.

  Second instance is named `I1`.

  Mosaic is named `M1` and mosaic instance (`mosaicInst`) is named `I2`.

  Third instance is named `I3`.

- For OpenAccess:

  First instance is named `I0`.

  Second instance is named `I1`.

  Mosaic is named `M1` and mosaic instance (mosaic = mosaicInst = OA arrayed instance) is also named `M1`.

  Third instance is named `I2`.

Binding of Mosaic Instances

In CDB, the master cells for mosaic instances are automatically bound (opened); in OpenAccess, the master cells for mosaic instances are not automatically bound. Therefore, for OpenAccess, applications and customer code must bind mosaic master cells explicitly.

For example, for the *`mosaicSwitcher`* argument of the `dbHierInfo` structure, you provide your own, user-defined mosaic switcher code, which you have registered. For OpenAccess, use the `dbGetAnyInstMaster` function to bind a mosaic.

The code example below works for both CDB and OpenAccess. For OpenAccess, the following line was added:

```
masterId = dbGetAnyInstMaster(info->id);
```

Code Example

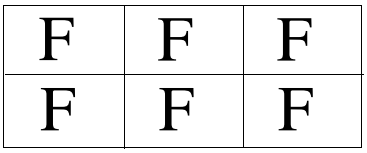
```
Boolean
myMosaicSwitcher( dbHierInfo *info )
{
    travArgs *a;
    a = (travArgs*)info->pSArg;
    mosaicSwitcher++;
    info->toExpand = FALSE;
    info->toProduce = FALSE;
    if(a->expand){
        dbCellViewId masterId;
        /****** Add the following line for OpenAccess: *****/
        masterId = dbGetAnyInstMaster(info->id);
        if( !a->baseMosaic ) info->toExpand = TRUE;
        if( a->baseMosaic == info->id ){
            info->toExpand = TRUE;
        }
    }
    return TRUE;
}
```

Rotation of Mosaics

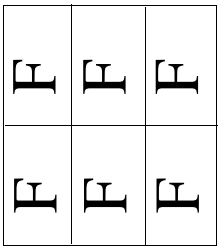
The difference between simple mosaics in CDB and emulated mosaics in the Virtuoso environment is the effect of rotation. For CDB, rotation is applied to the individual elements of the array, whereas in the Virtuoso Studio design environment, the rotation is applied to the complete array.

For example, for two arrays, one with angle 0 and the other with angle 90, the effect of rotation during creation is as follows:

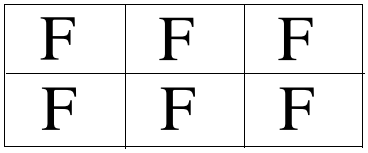
Zero degrees on CDB



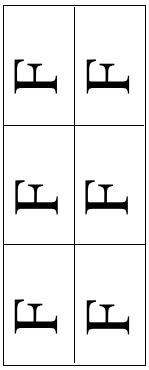
90 degrees on CDB



Zero degrees on OpenAccess



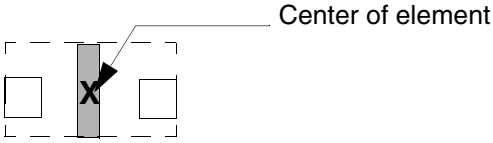
90 degrees on OpenAccess



Note: In CDB, the Property Editor swapped rows and columns to make the array appear as if it has been rotated, whereas this is not necessary in OpenAccess. OpenAccess completely rotates the rows and columns.

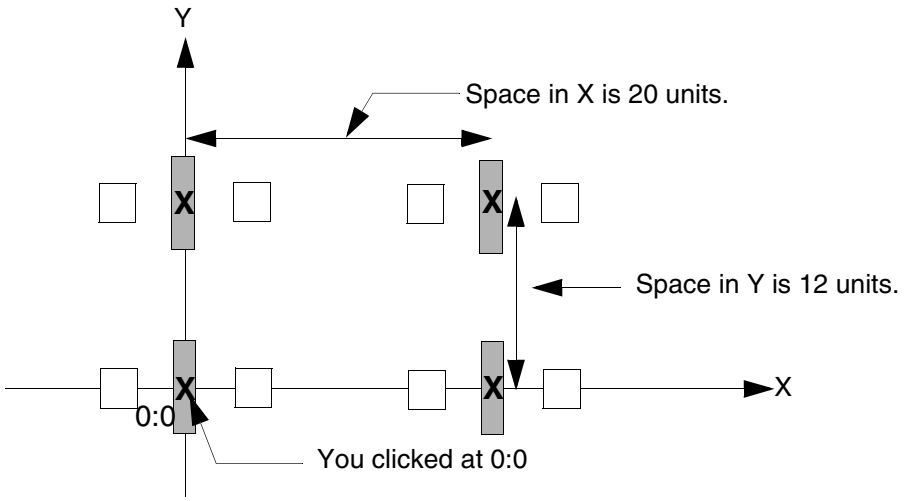
Therefore, the coordinates of the bounding box around a rotated mosaic instance are different for each database. Also, in CDB, the numbers in the Delta X and Delta Y fields on the Create Instance form were swapped.

In the following examples, the mosaic instance contains two rows and two columns of the following element:



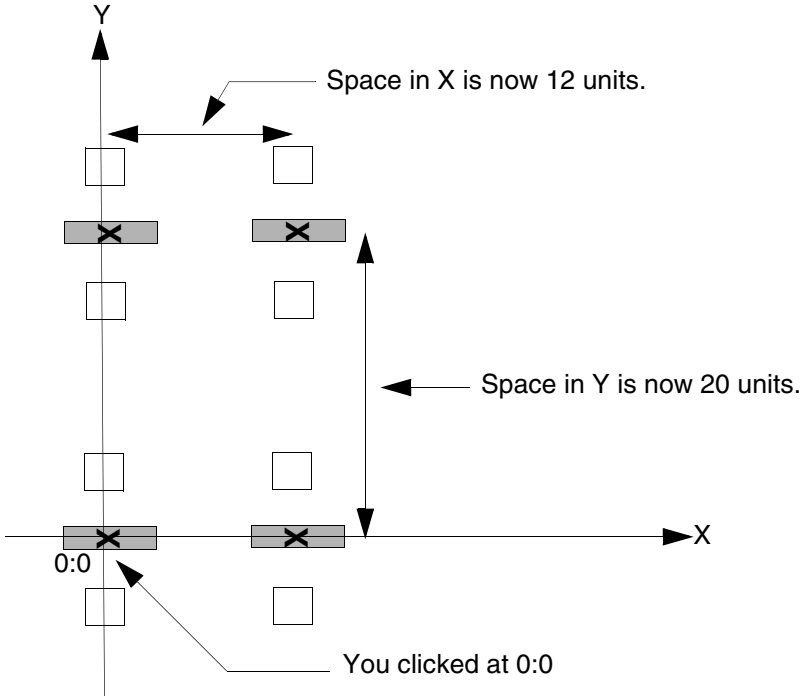
When you place the example mosaic instance without rotation, the results are the same for CDB and OpenAccess, as shown below. In the example, the space between the centers of the elements is 20 units along the X axis and 12 units along the Y axis. The centers of the left column of elements falls on the Y axis.

Figure 3-1 No Rotation: Results Are Same for OpenAccess and CDB



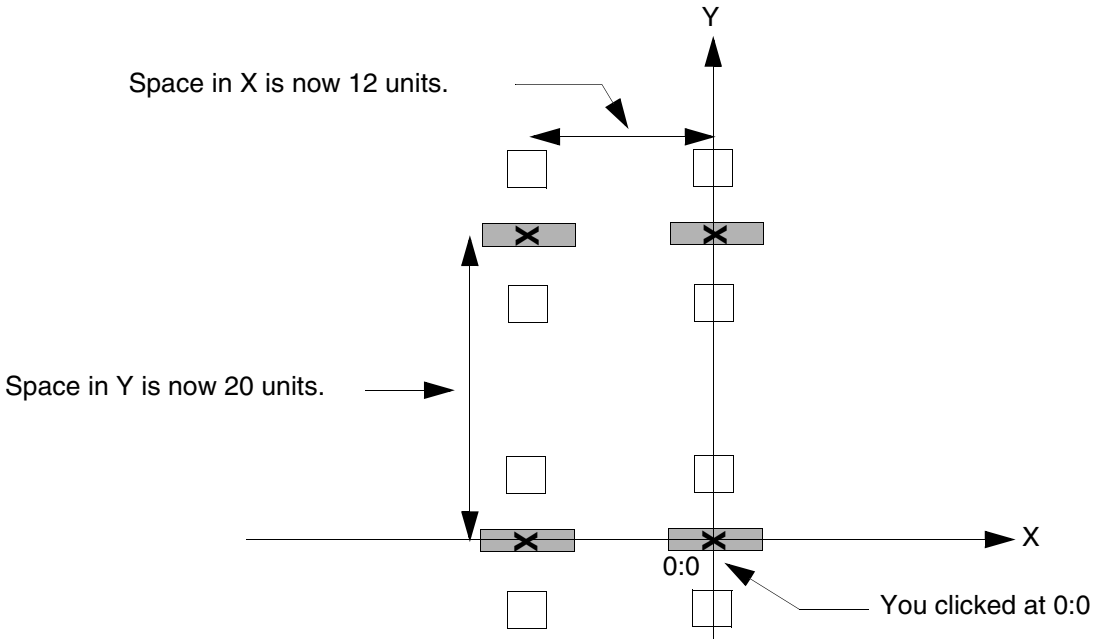
In CDB, when you placed the example mosaic instance with a 90 degree rotation, the space between the centers of the elements would change to 12 units along the X axis and 20 units along the Y axis, and the centers of the left column of elements would fall on the Y axis.

Figure 3-2 With Rotation: Location in CDB



In the Virtuoso environment, when you place the example mosaic instance with a 90 degree rotation, the centers of the right column of elements falls on the Y axis.

Figure 3-3 With Rotation: Location in OpenAccess



Also, `dbGetInstOrient` supports both regular and mosaic instances. On CDB, `dbGetInstOrient` supported only regular instances.

Design Management

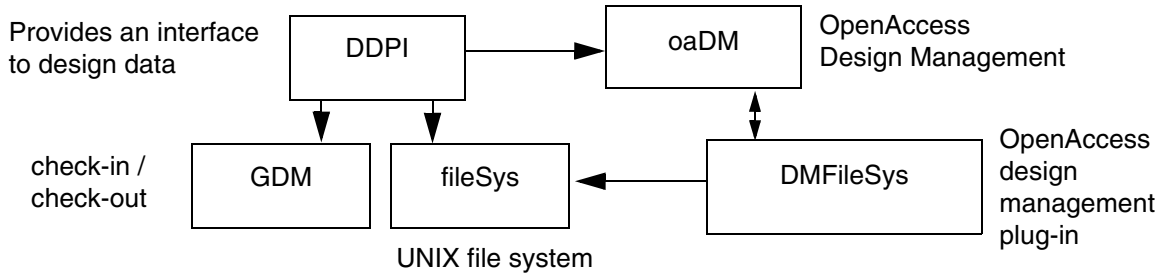
In OpenAccess the phrase “Design Management” refers to library management, work-space synchronization, file management, and version control management.

Different types of design management systems are supported using plug-in executables (the Virtuoso Studio design environment only supports the DMFileSys plug-in).

In the Virtuoso Studio design environment the phase “Design Management” refers to checkin and checkout type operations, which are handled by Cadence Generic Design Management (GDM). This is in contrast to OpenAccess where the term Design Management refers to all access, and to any object in the library (not just checkin an checkout.) In OpenAccess, there is a plug-in called Version Control which is the equivalent of GDM in the Virtuoso environment.

In the Virtuoso environment, GDM interfaces with applications, making direct design management system calls which handle version control (checkin and checkout). The data management system called Design Data Procedural Interface (DDPI) handles the low-level manipulation of the libraries, cells, and views. The mapping of OpenAccess design management to the Virtuoso environment is done through the DMFileSys plug-in which is implementing 5.X compatibility.

Figure 3-4 DMFileSys Interface with OpenAccess Design Management System



Differences in Storage and Data Models

Different types of OpenAccess design management plug-ins support different types of storage mechanisms. The Virtuoso Studio design environment relies on a specific storage mechanism that is consistent with the 5.X library structure implemented on OpenAccess with the DMFileSys plug-in.

The 5.X library structure leverages the UNIX directory structure. The overriding organization of library data is lib/cell/view. Each library is a separate UNIX directory. Each cell within a library is a separate file system directory. Each view within a cell is a separate file system directory. Each view directory contains application files. Files specific to a cell (and all views of that cell) can be placed in the cell directory. Files specific to a library can be placed in the library directory.

The data model in OpenAccess design management is slightly different than the data model in 5.X. The OpenAccess design management supports a cellview (combination of cell and view) that maps to a view in 5.X. The concepts of library and cell are the same. The concept of a *master* in the 5.X library structure maps to what is called a *primary* in OpenAccess design management.

The OpenAccess design management system is compatible with the Virtuoso environment through the DMFileSys plug-in. The 5.X storage structure is understood and maintained as well as cells and views are represented as directories.

For more information, see Design Management SKILL Functions and Design Management C Functions

Region Query Functions

In the Virtuoso Studio design environment the functions used to query regions have been re-implemented. A region query function has been provided for each type of the following objects; blockage, boundary, guide, instance, marker, row, shape steiner, and via. Also, the types of objects that are produced by these functions and the control actions have been changed.

For more information, see Query SKILL Functions and Query C Functions.

Important

Query of objects is based on the open hierarchy level. Only cellviews that are currently open will be searched. To ensure that all shapes are returned in a query list, `dbOpenHier` can be used to load all required cellviews.

Changes to C Functions for Region Query

The procedural interface function, `dbProduceOverlap2` produces figures within a given search box. In the Virtuoso environment, the functions that produced figures in a given searching box will only produce figures that overlap the area specified by the searching box.

In CDB, the `dbProduceOverlap2` function can be set to either; only produce figures contained within a specified search box Figure 3-5 on page 68, or produce any figure whose bounding box overlaps the specified search box Figure 3-6 on page 68. This behavior in CDB is controlled by setting the `trueOverlap` member of the `dbHierInfo` structure which is passed to `dbProduceOverlap2`.

In the Virtuoso environment, `dbProduceOverlap2` will only produce figures contained within the specified search box Figure 3-5 on page 68. This behavior is know as producing true overlaps.

The true overlap behavior in CDB only applied to the line, path, and polygon shapes. In the Virtuoso environment, true overlaps applies to all figures and not just these three shapes.

Figure 3-5

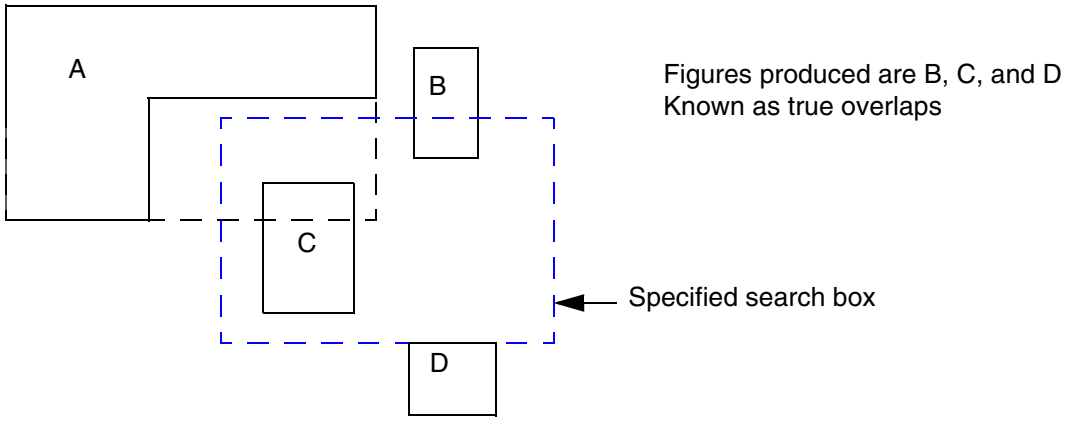
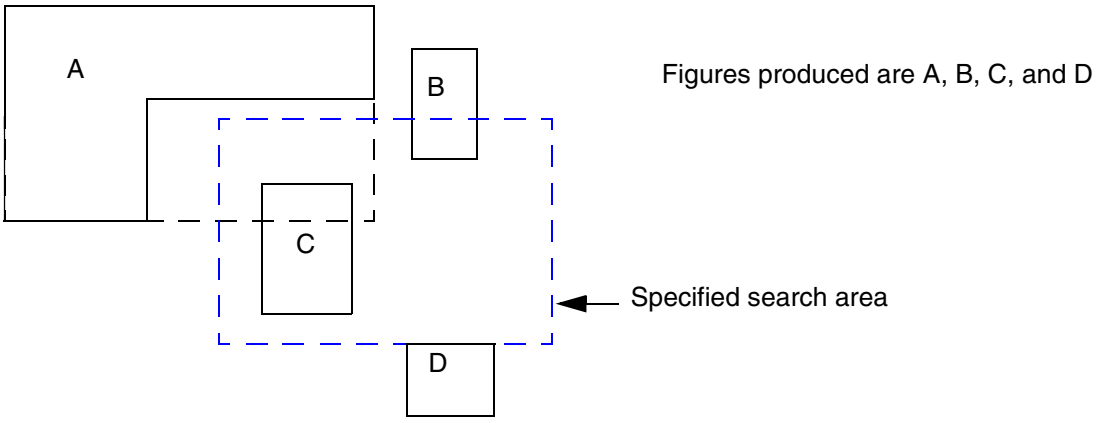


Figure 3-6



Changes to SKILL Functions for Region Query

The SKILL functions; `dbGetTrueOverlaps`, `dbGetOverlaps`, and `dbProduceOverlap`. have been changed.

In CDB the SKILL function `dbGetTrueOverlaps`, which returns a list of figures within a specified search box, is applicable to only lines, paths, and polygons. In the Virtuoso environment, all shapes are applicable.

In CDB the SKILL functions `dbGetOverlaps` and `dbProduceOverlap` return a list of all figures whose bounding box overlaps the specified search box. In the Virtuoso environment, `dbGetOverlaps` and `dbProduceOverlap` will produce the same results as `dbGetTrueOverlaps` (Figure 3-1).

For more information, see Query SKILL Functions and Query C Functions.

Placement Status

The following describes the placement status supported in the Virtuoso Studio design environment.

Placement Status	Description
`none`	The placement status has not been set, but the current placement location is treated as valid. The value for this placement status retrieved by `instId~>status` is `nil`.
`unplaced`	The object is defined logically in database, but not physically placed in the design. The current placement is arbitrary and should be changed. Unplaced objects do not contribute to the bounding box of the containing design, and they will not be returned by region query.
`fixed`	The placement of the object can not be changed by an automatic placement tool
`locked`	The placement of the object can not be changed
`placed`	An effort has been made to place this object but it may be placed again by future attempts

In CDB, the status of `firm` was the default placement status and was used to represent an object should not be changed by an automatic placement tool. On OpenAccess, the default placement status is none.

Note: *Design Summary – Instance Statistics* displays the number of instances with the placement status attribute set to `unplaced`. This attribute is maintained by P&R applications. An `unplaced` instance is not displayed in the Virtuoso layout window, it is however present in the database.

The following describes the placement status mapping from CDB to OpenAccess.

Non-Preview Data

CDB	OpenAccess
firm	none
suggested	placed
placed	placed
locked	locked

Preview Data

CDB	OpenAccess
firm	fixed
suggested	placed
placed	placed
locked	locked
unplaced	unplaced

Virtuoso Studio design environment Libraries

Determining the Library Structure

When a library is created, the library structure is determined by which plug-in is used. The Virtuoso environment only supports the DMFileSys plug-in. In order to be understood as a

5.X library, the library must be defined as a 5.X library in the library definition. Any library defined in a `cds.lib` file is assumed to be a 5.X library.

The `ddLibIs5X` C and SKILL functions determine if a library was created with 5.X library structure or not. `ddLibIs5X(ddId libId)` uses a library ID as an argument and returns TRUE if the library was created as a 5.X library otherwise it returns FALSE.

Using Nested Libraries

Although both CDB and OpenAccess allow hierarchical library structures to be defined, using nested libraries (libraries within other libraries) can cause unpredictable behavior and cryptic errors in applications. OpenAccess issues warnings about nested libraries; CDB does not.

In OpenAccess, there are a number of conditions that result in warnings related to nested libraries. For example, when a library list file containing nested libraries is read, OpenAccess issues a warning saying that the directory has been used multiple times.

Library Definition Files

In CDB, the `cds.lib` is the only library definition file used by applications. In the OpenAccess Virtuoso Studio design environment, the `cds.lib` is the only supported library definition file. The CdsLib plugin (release 31.09) allows OpenAccess applications to read `cds.lib` files, where previously they required to use `lib.defs`.

Combining CDB and OpenAccess Libraries

Cellviews containing `.oa` files must not coexist in libraries whose cellviews contain `.cdb` files. Similarly, CDB libraries can contain a `techfile.cds` file and Virtuoso Studio design environment libraries can contain a `tech.db` file, but not a combination of these types of files.

Cellviews

Creating Cellviews

In the Virtuoso Studio design environment, a cellview is not created unless a master (`layout.oa`, `sch.oa`, `verilog.v` etc.) has been specified. For example:

```
ddGetObj(.... "verilog.v", "w")
dbOpenCellViewByType(.... "netlist" "wc")
```

The above would results in `verilog.v` as the master and the `netlist.oa` as a separate file.

```
dbOpenCellViewByType(.... "netlist" "w")
ddGetObj(.... "verilog.v" "wc")
```

The above would results in `netlist.oa` as the master and `verilog.v` as a separate file.

For more information, see ddGetObj.

Saving Cellviews Using SKILL

You can edit the contents of a cellview with SKILL functions in the CIW or by running SKILL programs, but you cannot save your edits while in read (r) mode. To save edits made with SKILL, change to edit mode either before or after making the edits to the cellview.

Panic Cellviews

Panic cellviews are supported in the Virtuoso Studio design environment. In addition, the SKILL functions, `dbOpenPanicCellView` and `dbOpenAutoSavedCellView` are also supported.

Scratch Cellviews

CDB provided a scratch cellview mode, `"s"`, that let you create temporary cellviews in virtual memory; you could edit them, but not save them to disk. In the Virtuoso environment, you need to use the editable read-only capability instead. See Cellview Reference Count on page 77.

On CDB, the following SKILL example created a scratch cellview:

```
cvId = dbOpenCellViewByType( "myLib" "myCell" "myView" "maskLayout" "s" )
```

The Virtuoso environment provides editable read-only cellviews that cannot be saved to disk; therefore, scratch cellviews are redundant and are being retired (with the exception of Pcell submasters, see Pcell Submaster are Opened in "s" Mode on page 73).

The following SKILL code creates a cellview that provides the same functionality as a scratch cellview on CDB:

```
cvId = dbOpenCellViewByType( "myLib" "myCell" "myView" "maskLayout" "w" )
dbReopen( cvId "r" )
```

Important

SKILL code that explicitly uses the `"s"` scratch cellview mode needs to be updated to use the editable read-only mode, as shown above, when running on OpenAccess.

Scratch Cellviews Can Have Null Library in OpenAccess

On CDB, scratch cellviews were associated with a specific library; in the Virtuoso environment, the equivalent of scratch cellviews, editable read-only cellviews, are not associated with a library.

In the Virtuoso environment, standard vias (`stdVia`) do not have an associated library. The data translated from LEF In or DEF contain routes and standard vias, and therefore, master cellviews for standard vias that do not have libraries.

If you have SKILL or C code that requires the library for a cellview, you must to update your code. For example, `dbGetOpenCellViews` returns a list of all open cellviews, which could include scratch cellviews without associated libraries.

Currently, the absence of an associated library only applies to standard vias. The Virtuoso Studio design environment provides the SKILL API `dbIsCellViewStdViaMaster` to determine whether a cellview is a stdViaMaster.

Pcell Submaster are Opened in "s" Mode

Scratch mode, "s", still exists for Pcell submasters. Pcell submasters now are opened in "s" mode whereas they were opened in "a" mode on CDB. The "s" mode is for internal use only and cannot be set by the user.

Code that checks for write access can check the mode as follows:

```
mode!="r"
```

Standard Via Masters

In the Virtuoso Studio design environment, the masters of standard vias are not associated with a library, cell, or view name. A NULL value is returned when attempting to use `dd` function calls to return the master ID of standard vias. The SKILL function `dbIsCellViewStdViaMaster` is provided to determine whether a cellview is a standard via master or not.

For more information, see Using Function Calls on Standard Vias on page 36.

Behavior in Read-Only Mode

- CDB and the Virtuoso Studio design environment respond differently when you switch from edit to read-only mode using the `dbReopen` function.

  In CDB, when you open a cellview in edit mode, edit the cellview, and then change to read mode using the `dbReopen` function, your changes are retained in virtual memory (VM). When the edited cellview is purged from VM or you exit the editing session, the changes are lost. You can preserve your changes by saving the cellview under a new name (Save As).

  In Virtuoso Studio design environment, when the `dbReopen` function is used to change from edit to read-only mode, the changes are discarded immediately.

In both CDB and Virtuoso Studio design environment, when the graphical user interface is used to change from edit to read-only mode, the system displays a dialog box asking if you want to save your changes.

- CDB and the Virtuoso Studio design environment respond differently when you delete an object from a read-only cellview.

  In CDB, you cannot modify or delete an object from a read-only cellview.

  In the Virtuoso Studio design environment, you can delete an object from a read-only cellview by using the geDeleteSelSet function.

  Note: This action cannot be undone.

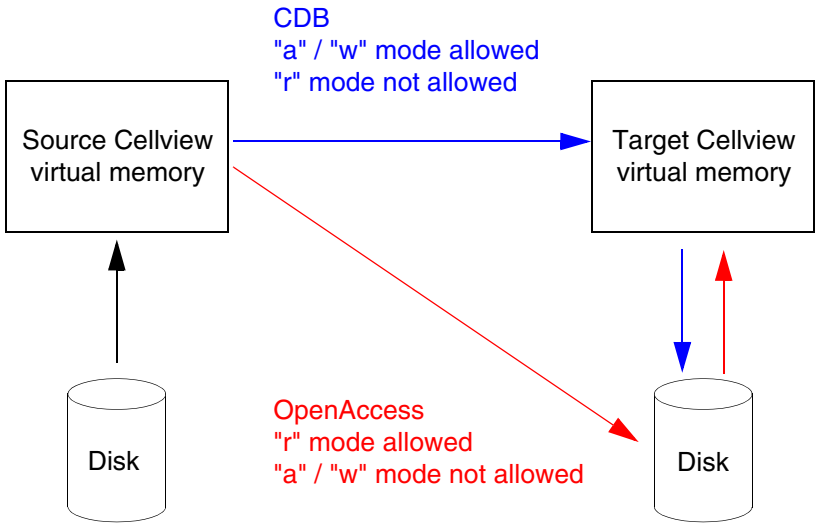
Tip

If you want to prevent deletion of objects from a read-only cellview, use the leHiDelete function. You will be warned to open the cellview with write permission.

Tip

If you want to prevent deletion of objects from a read-only schematic cellview, use the schHiDelete function. If you try to delete the object, the SCH-1007 message will be displayed.

Using dbWriteCellView and dbCopyCellview



OpenAccess checks to see if the cellview is open in virtual memory. If the cellview is open in "a" or "w" modes in virtual memory, you can not "save as" or copy to the cellview. This is to prevent the disk copy of the cellview and the copy in virtual memory from getting out of sync.
if the cellview is open in read mode, the copy on disk is updated and the copy in virtual memory is updated.

When saving a cellview using the command *Design – Save As*, the target cellview cannot be open in virtual memory. When attempting to use the *Save As* command on a cellview that is open, the cellview is considered a locked file. The *Save As* command is designed to be used to save a source cellview to a different target cellview without overwriting the original.

When using the functions `dbWriteCellView` or `dbCopyCellView` the following warning message will be issued when attempting to “save as” or copy to a cellview which is open in virtual memory in append (`a`), or write (`w`) mode.

```
*WARNING* dbWriteCellView: Unable to lock database file for library/cell/view
```

If the target cellview is open in read (`r`) mode, the target cellview is saved to disk and updated in virtual memory.

CDB allowed the target cellview to be open in append (`a`), or write (`w`) mode when using when using `dbWriteCellView` or `dbCopyCellView`. The target cellview was updated in virtual memory but not saved to disk.

Opening a Cellview with dbGetAnyInstSwitchMaster

The Virtuoso Studio design environment behaves differently than CDB when choosing which view to open based on an instance’s switch master.

- In CDB, the `dbGetAnyInstSwitchMaster` function uses the instance's library and cell names to access the list of view names, and opens the first view that contains CDB data; the function returns a handle to the cellview. In addition, for Pcells, the system uses each parameter on the instance to check whether it is a parameter of the matching cellview, and returns a handle to a submaster.
- The Virtuoso Studio design environment opens the first view in the list of views that matches. If the matching view does not contain OpenAccess data, the system issues a warning message and the function returns NULL. The list of views is not searched for a view containing OpenAccess data; it stops when it finds the first view in the list of view names on disk.

This behavior change makes the Virtuoso Studio design environment consistent with other applications that could be using the Hierarchy database and Hierarchy Editor to access the list of view names. In a hierarchical traversal of data, the first view matching in the view name list would be used regardless of the data stored in it.

Maximum Number of Open Files

OpenAccess avoids problems caused when other processes write to a file that has been partially read by keeping partially read files open. The maximum number of open files is limited by the operating system: in Solaris 7, the maximum number of open files is 1024. You can set the number from 1 to the maximum of 1024 with the following UNIX shell command:

```
limit descriptors your_maximum
```

For example, to set the maximum number of open files to 512, type:

```
limit descriptors 512
```

To reset the number to the maximum of 1024, type:

```
unlimit descriptors
```

Refer to the OpenAccess documentation for the minimum number of descriptors expected by OpenAccess. Currently, the minimum limit is 512. Do not use less than the minimum, as the system might not behave as expected.

OpenAccess always completely reads small files, and partially reads large files, as long as the total number of open files does not exceed the current limit. When the number of open files would exceed the limit, OpenAccess completely reads large files. The intention is to partially read large files whenever possible.

Cellview Reference Count

CDB does not close instance masters even when there are no instances referring to them. OpenAccess purges the instance master as soon as the reference count associated with the instance master becomes zero. This happens when no open cellview has an instance of the master.

Frequent opening and closing of instance masters adversely affects the performance of Virtuoso tools. To avoid this problem, Virtuoso increments the cellview reference count twice when a cellview is explicitly opened.

Note: Pcell supermasters and scratch cellviews are an exception; the Virtuoso environment does not increment twice for Pcell supermasters and scratch cellviews.

The `dbGetRefCount` C function has been updated as follows: When the OpenAccess reference count is two (2) or greater, one (-1) is subtracted from the OpenAccess reference count. The function `dbGetRefCount` now returns a different count than the native OpenAccess `oaCellView::getRefCount()` function: when the reference counter is greater than zero (0), the OpenAccess function count is higher by one (1).

For the Virtuoso Studio design environment, the C and SKILL `dbClose` SKILL functions have been modified to close the cellview twice when the reference count is two (2), thereby removing the cellview from virtual memory.

CellViewType Attribute

In CDB, the cellview attribute `cellViewType` could be modified. In OpenAccess, the `cellViewType` attribute cannot be modified. The `cellViewType` attribute must be set to its correct value when the cellview is initially opened for write access.

Mapping of Cellview Types from CDB to OpenAccess

There are several differences in the support of view types for CDB and OpenAccess, the most important of which is that OpenAccess supports fewer view types than CDB. View types are mapped to one of the four OpenAccess view types, as shown in Mapping of CDB View Types to OpenAccess View Types.

- OpenAccess does not use the data registry; it uses its own filename-to-view type mapping, which is different from the mapping defined in the data registry.
- OpenAccess enforces the built-in mapping from filename-to-view type; it does not allow data filenames that do not match the view type stored internally in the data.

- The `dbOpenCellViewByType` function maps all CDB view types to one of the four OpenAccess view types.
- The `stranger` view type is not supported.

The mapping of view types between CDB and OpenAccess if given in the table below.

Table 3-1 Mapping of CDB View Types to OpenAccess View Types

CDB enum	OA enum	CDB Filename	OA Filename	CDB View Type	OA View Type
dbcUnknownCellViewType	(no equivalent)	N/A	N/A	N/A	N/A
dbcGraphic	oacMaskLayout	graphic.cdb	layout.oa	graphic	maskLayout
dbcMaskLayout	oacMaskLayout	layout.cdb	layout.oa	maskLayout	maskLayout
dbcSchematic	oacSchematic	sch.cdb	sch.oa	schematic	schematic
dbcSymbolic	oacMaskLayout	layout.cdb	layout.oa	maskLayout	maskLayout
dbcPCB	oacNetlist	N/A	netlist.oa	N/A	netlist
dbcSchematicSymbol	oacSchematicSymbol	symbol.cdb	symbol.oa	schematicSymbol	schematicSymbol
dbcLogicModel	oacNetlist	N/A	netlist.oa	N/A	netlist
dbcBehavioral	oacNetlist	N/A	netlist.oa	N/A	netlist
dbcStranger	(no equivalent)	stranger.cdb	N/A	stranger	N/A
dbcNetlist	oacNetlist	netlist.cdb	netlist.oa	netlist	netlist
dbcPackage	oacNetlist	N/A	netlist.oa	N/A	netlist
dbcVerilogMap	oacNetlist	veriMap.cdb	netlist.oa	verilogMap	netlist

Custom Virtuoso Studio design environment View Types

The Virtuoso environment supports the creation of custom, user-defined view types through the data registry. You can store and manage any non-design data (non-dbObject) in cellviews in your design library for use by third-party (non-Cadence) applications within the Cadence 5.x architecture library structure. A typical use for custom user-defined view types is to store files or documentation in ASCII text, PDF, or HTML format.

Customer code using registered, non-design, user-defined is supported. However, code using user-defined view types must not use database calls, such as `dpOpenCellViewByType`.

Important

Make sure that code using (reading/writing) user-defined view types does not contain any Virtuoso or OpenAccess database calls (SKILL API, C API (ITK/DB), or native OpenAccess API).

The Virtuoso environment introduces no new restrictions for user-defined view types used to store non-OpenAccess data. Follow the standard documented procedures to register a tool, data format, and view alias for your user-defined view type in a `data.reg` file. For more information, see "Cadence Data Registry File: data.reg" in the *Cadence Application Infrastructure User Guide*.

Properties of Libraries, Cells, and Views

In CDB, properties of libraries, cells and views are stored in files called `prop.xx`. To access the properties, the application would call `dbOpenBag`. The property bag is explicitly or implicitly opened to create properties on library, cell or view.

In the Virtuoso environment, when using the DMFileSys system, the `prop.xx` file has been replaced by a `data.dm` file. In OpenAccess the properties of objects such as library, cell or view are created directly in this file and accessed through the corresponding objects. In the Virtuoso environment, the emulation code provides access to this information through the emulate property bag SKILL and C functions.

Calls to dbBag SKILL Functions

In CDB, SKILL functions such `dbReplaceProp` would automatically open the property bag of a database object. In the Virtuoso environment, applications calling functions must call `dbOpenBag` first, then `dbSaveBag` and `dbCloseBag` when all property modifications have been completed.

When using a SKILL API such as `dbGetPropByName` on a `dd` ID when the property bag is not open, the function fails and issues a warning message.

The mode argument for `dbOpenBag` differs for CDB versus OpenAccess. For more information, see dbOpenBag in the Appendix A.

Property Type dbcFileNameType

One CDB property type, `dbcFileNameType`, is not supported. The translator issues a warning on encountering this property and converts it to `dbcStringType`.

Properties versus Parameterized Cell Parameters

CDB implemented parameters as properties, which implies that a parameter has an associated ID. In OpenAccess, parameters are objects, which while being distinct from properties, are not first class objects (an OpenAccess parameter object does not have an ID). For applications and customer SKILL code to work as before, the CDBA procedural interface layer produces IDs for parameters as if they were properties. A `dbProp` class creates a filter to determine whether a property, based on original perspective, represents an OpenAccess property or an OpenAccess parameter, and to act appropriately.

In CDB, you must not change the value of a parameter for a Pcell supermaster using SKILL because the submasters will not be properly updated.

Note: In this context, parameters means Pcell parameters, not CDF parameters. There is no change to CDF parameters. The Virtuoso environment supports ranges for parameters, but does not support enum types for parameters. OpenAccess itself does not support ranges or enum types for parameters.

Semantics of an Instance Property that Matches the Supermaster Parameter Default Value

On OpenAccess, when a Pcell supermaster is recompiled, default parameter values may change. Instances that have explicit properties that match the old supermaster parameter default values retain their original values. Instances that inherit the property values from the supermaster are updated to reflect the new supermaster defaults.

The complete set of properties explicitly set on the instance determines its binding to the Pcell submaster. An instance that has a single property whose value matches the parameter value in the supermaster binds to a different submaster than an instance that has no explicit properties and which inherits all of the parameter values from the supermaster, even though both instances appear to have the same value for the property.

Note: The *Edit Properties form* and the *Property Editor* assistant can be used to modify instance properties that correspond to supermaster parameters (and affect the underlying Pcell subMaster accordingly). However, by using these tools, you cannot create an instance property that duplicates a supermaster parameter; the new property must have a value that differs from the superMaster parameter value.

Command Interpreter Window (CIW)

For information about CIW user preferences, see “Specifying CIW Controls” in Chapter 8, of the *Virtuoso Design Environment User Guide*.

For information about CIW control environment variables, see “Specifying CIW Controls in the .cdsenv" in Chapter 9, of the *Virtuoso Design Environment User Guide*.

Removed User Preference Environment Variables

The environment variables `formButtonLocation` and `textFont` are no longer used by Virtuoso applications.

Objects

The following sections describe differences when modifying objects for CDB and OpenAccess.

Changing Object Types

CDB supported many functions that would convert an object from one type to another. For example, `dbConvertPathToPolygon`, `dbConvertEllipseToPolygon`, and `dbConvertLineToPath`. In CDB, the object identifier returned by the specific conversion function was the same as the object identifier that was passed in. However, the CDBA layer cannot guarantee this behavior; the object identifier returned might be different from what was passed in. `dbConvertLineToPath` is one function in particular where the path identifier returned will be different from the line identifier the function was given.

Another function that will change the underlying database object is `dbSetLabelType`. A normal label in CDB is equivalent to an OpenAccess `oaText`. However, an IL or NLP label is equivalent to an OpenAccess `oaEvalText`. In CDB, the `dbSetLabelType` function returned a Boolean indicating whether the operation was successful or not; in OpenAccess, `dbSetLabelType` returns an ID, which changes when the label type changes. Applications calling this function should be reviewed to see whether they refer to the given identifier argument after calling it.

Changing the Name of an Object

In CDB, the name of an object is merely another attribute of the object. Therefore, changing the name of an object does not change the object or its resulting identifier. This is true even when the new name implies a different number of bits than the old name.

In the Virtuoso environment, changing the name of an object might result in a different object. For example, if the name of an instance is changed from `I10<0:4>` to `I10`, the result is replacing the vector instance named `I10<0:4>` with a scalar instance `I10`. Many of the

CDBA procedural interface functions return a Boolean, indicating only whether the operation was successful or not. Applications that call functions which set the name of an object must be reviewed in case they rely on the object identifier to remain the same.

Some functions, including `dbSetTermName`, `dbSetInstName`, `dbSetAnyInstName`, and `dbSetAnyInstNumInst`, have been modified to return an ID rather than just a Boolean value. The `dbSetTermName` function has the restriction that the new name must imply the same number of bits as the net that is associated with the terminal. There are cases where the ID will be same, but applications that cache the identifier should be reviewed for their use of these functions.

Changing an Instance Name to the Same Name

The Virtuoso environment and CDB database behave differently when you change the name of an instance to its existing name.

- The Virtuoso environment checks to see whether an instance already exists with the same name, and finding that it does, issues the following warning message:

  ```
  *WARNING* (dbSetAnyInstName) Instance with name 'instance_name' already exists
  ```

- CDB did not check to see whether an instance already exists with the same name. The name would be replaced with the identical name and no warning message was issued.

Timestamps

For timestamps, the primary difference between CDB and the Virtuoso Studio design environment is that CDB used the `instancesLastChanged` property to save the time that a cellview was updated, while the Virtuoso environment uses counter objects (such as `oaTimeStamp`), and increments them when their associated objects are updated.

How CDB Uses Timestamp Properties

CDB maintained several time properties in the database, the most prominent of which was called `instancesLastChanged`. This property was updated every time a modification to the database was made.

CDB and all of its connectivity extractors, including Virtuoso XL and the Virtuoso Schematic Editor, used the following timestamp properties:

- `instancesLastChanged`: This property was maintained solely by CDB and updated whenever an edit was made to a cellview. All applications use this property to determine whether the database has been edited.

Note: Despite the name of the `instancesLastChanged` property, this property contains the time of the last modification of the contents of the cellview, not its instance masters.

- `lastSchematicExtraction`: To designate that the cellview had been extracted, and thus the connectivity data had been created or updated, extractors placed this cellview property in the database. The database stored two major types of data: graphical data and connectivity data. For a schematic cellview, the connectivity was extracted from the graphical data by the Virtuoso Schematic Editor extractor.

To determine whether the connectivity of a cellview was current, applications compared the timestamp property `lastSchematicExtraction` against the timestamp property `instancesLastChanged`. If the value of `lastSchematicExtraction` was later than that of `instancesLastChanged`, the connectivity data was current.

How the Virtuoso Studio design environment Uses Counters

In the Virtuoso Studio design environment, the method for tracking whether the database has been edited is by updating unsigned integer counters.

Note: Although counters in OpenAccess are called “timestamps,” their values are unsigned integers.

In addition to maintaining a counter for each cellview, the Virtuoso environment maintains counters for many other database objects. Whenever an edit is made the counter increments the cellview by 1 and the counter for the object by 1.

Counters Provide Finer Resolution

An advantage of counters over timestamp properties is finer resolution. Time values are calibrated to one-second intervals. Where an edit and a “Check and Save” are completed quickly, the values of the `instancesLastChanged` and `lastSchematicExtraction` properties can remain the same, even through the cellview and connectivity data had changed. This stymied applications, particularly incremental netlisting, which relies on the comparison of these properties as a preliminary test to control whether the cellview needs to be re-netlisted. Using counters prevents this problem because counter values are increased by 1 as soon as an edit is made.

Instance Time Stamps

Virtuoso applications determine that connectivity data is up to date using the cellview counter maintained by OpenAccess. The cellview counter is the OpenAccess equivalent of the CDB

property `instancesLastChanged`. The following new database C and SKILL APIs have been supplied for this purpose:

- In C:

```
Boolean
     dbIsConnCurrent( dbCellViewId cellViewID )
```

- In SKILL:

```
dbIsConnCurrent( d_cellViewId )
=> t | nil
```

At the end of the extraction process, the Virtuoso Schematic Editor extractor creates a signed integer cellview property named `connectivityLastUpdated`. This is the OpenAccess equivalent to the CDB property `lastSchematicExtraction`.

To tell whether connectivity data is up to date, applications compare the value of the cellview counter against the value of the `connectivityLastUpdated` property. If the value of the cellview counter is not equal to the value of `connectivityLastUpdated`, the connectivity data is out of date and needs to be re-extracted.

Currency of an Instance and Its Master

After an instance is placed, its master could change. Important examples of changes include swapping pin locations and adding, deleting, or renaming pins. As masters can be instantiated in many designs residing in libraries separate from the libraries containing the masters, users need to be reminded of changes to masters. The Virtuoso Schematic Editor does this by placing blinking yellow markers on instances having changed masters when the schematic is opened (and only when the schematic is opened). It does this as follows:

- In CDB: When a schematic was opened, the value of the `instancesLastChanged` property was retrieved from the master during instance binding. This value was compared to the value of the `instancesLastChanged` property from the schematic. If the value from the master was later than the value from the schematic, the master had been edited. When the master had been edited, the Virtuoso Schematic Editor placed blinking yellow markers on the instances. Blinking yellow markers might have also indicated that the schematic needs to be re-extracted to update connectivity.
- In the Virtuoso environment: the value of the cellview counter from the master is stored in the instHeader for instances, under the name `masterChangeCount`. When the schematic is opened again, the cellview counter from the master is compared to the `masterChangeCount` counter from the instHeader. If the values are not equal, then the master has been edited; the Schematic Editor places blinking yellow markers on the instances.

Both Counters and Timestamps Needed in Some Applications

For some applications, like incremental netlisting, using counters to track what is up to date is not enough; these applications need to use file timestamps as well. The result is that after migration, while the extraction is up to date, the netlister will probably consider the design as modified and do a full re-netlisting, as opposed to an incremental run.

4

OpenAccess Technology Data Changes

This chapter describes the mapping between OpenAccess and Virtuoso for your technology data:

OpenAccess Data Models

With every new version release (data model revision), OpenAccess adds new kinds of data to the previous data model. The following table lists the prominent features associated with each data model revision.

Data Model Revision	OpenAccess Version	Feature
0	2.2.0 through 2.2.3 (p032)	Base OpenAccess 2.2 data model
1	2.2.4 (p033)	Huge databases
2	2.2.6 (p044)	FigGroups, constraint features
3	2.2.6 (p044)	Incremental Technology Database
4	22.04.001	45/65nm constraints GroupDef ConstraintGroupDef Pre-Defined Via Parameters Constraint Group Operators Purpose-Aware Constraints Reserved Purposes ITDB Constructs Constraint Parameters
5	22.50.001	Huge String/Name Data
6	22.60.005	oaPartitions Huge oaPointArray Data Huge oaAppProp Data

Virtuoso Support for OpenAccess Data Models

This section describes the features of OpenAccess and The table below maps the Virtuoso support for OpenAccess data models in the various IC releases.

IC Virtuoso Release	OpenAccess Data Model
IC6.1	0, 1, 2, 3
IC6.1.1	0, 1, 2, 3
IC6.1.2	0, 1, 2, 3, 4
IC6.1.3	0, 1, 2, 3, 4
IC6.1.4	0, 1, 2, 3, 4
IC6.1.5 (ICADV12.1)	0, 1, 2, 3, 4
IC6.1.6 (ICADV12.2)	0, 1, 2, 3, 4, 5
IC6.1.7 (ICADV12.3)	0, 1, 2, 3, 4, 5
IC6.1.8 (ICADVM18.1)	0, 1, 2, 3, 4, 5, 6

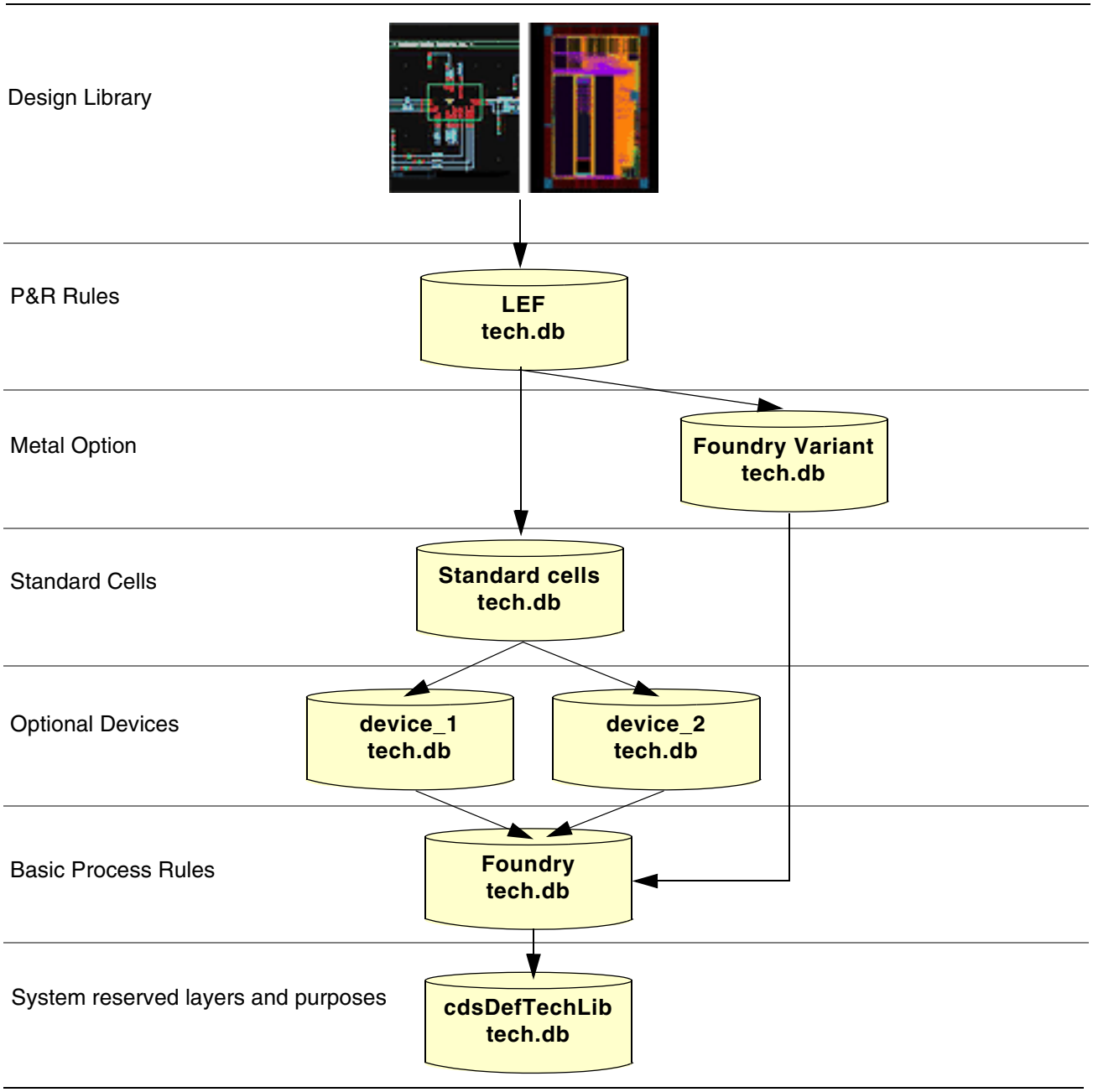
Support for Incremental Technology Database

IC6.1.x releases see the introduction of incremental technology databases (ITDB). This method of handling technology data allows technology information from multiple sources, such as the foundry, an IP provider, or designer at different times in the design cycle. Applications can incrementally assemble technology information by creating references from one technology database to other technology databases. This will allow the appropriate design team the rights to manage only the specific section of technology information that is relevant to them.

In the following example, the LEF technology database, referenced by the Design Library, references two technology databases, the Foundry Variant technology database and the Standard Cell technology database. This means that the design constraints and process information in the referenced databases are applicable to the Design Library design. The Design libraries technology database is a derived technology database because it is, in effect, inheriting information from the technology databases that it references. Furthermore, both

device technology databases reference the Foundry technology database, which references the default technology database. This chain of references is known as the technology database graph.

Each reference is created separately, and the technology database graph is created incrementally.



Setup

If incremental technology databases are not applicable to your design needs, your current technology database requires no setup or modification. The software, however, now handles all technology databases as incremental technology databases. Consequently, it will automatically remove system-reserved layers and purposes from your technology database and add a reference to *cdsDefTechFile*, the default technology database that defines system-reserved layers and purposes.

Conflict Checking

Since a technology library can inherit information from other technology libraries through an ordered set of references, conflicting objects are not allowed in a technology database graph. When a technology file is compiled, checks are done to determine if the objects with the same matching characteristics already exist within the database graph.

New Technology File Subsection

A technology file section has been added which lists the ordered technology libraries. For example,

```
controls(
   techParams( ... )
   viewTypeUnits( ... )
   mfgGridResolution( ... )
   refTechLibs(
      "StdCells"
      "PRtech"
      "Foundry"
   )
)
```

For information, see the *Virtuoso Technology Data User Guide*.

Constraint Support for 45/65 nm Process Rules

From IC6.1.2, Virtuoso leveraged the new OpenAccess data model 4 constraints to support the constraints for 45 nm and 65 nm processes for constructing correct layouts and to support the new LEF constructs.

Constraint Support for 32 nm Process Rules

From IC6.1.4, Virtuoso leveraged new constraints that have been added to OpenAccess data model 4 to support the 32 nm foundry design rules. Some existing constraints have also been

enhanced to provide support for 32 nm processes. The new constraints and constraint options also support the new LEF 5.8 constructs. The new constraints are Cadence-specific and therefore, the prefix `cdc` is used for those constraints.

Note: When searching for the constraint by name, drop the `cdc` prefix and change the initial letter change to lower case. For example, for `cdcCutClass`, use `cutClass`.

For writing a technology file that uses the new constraints, you need to start from a previous technology file from the same foundry, such as use 65 nm technology file to write a 45 nm technology file.

Support for Using Group Definitions

The oaGroupDef allows you to specify the definiton of a group. You can restrict the membership of the group by defining the object types that can be placed in the group as well as the databases in which the group can be created. For example, you can define a group that contains only oaNets.

Virtuoso leverages some of the OpenAccess built-in groups definitions for grouping nets. You can define the net groups using the Virtuoso Constraint Manager. You can create process constraints on these net groups using the Process Rule Editor.Using the Process Rule Editor, the process rules can be created either in the design or in the technology database.

ConstraintGroupDef

An `oaConstraintGroupDef` is a public object that specifies a definition for a constraint group. The definition contains a name, a type, a list of databases that the constraint group can be created in, a list of objects (by type) that the constraint applies to, a relationship type, and an indicator of whether or not there can be only one constraint group that uses this definition in any given database.

There are a set of built-in `ConstraintGroupDefs` with associated semantics:

- `oaImplicit`
- `oaDefault`
- `oaFoundry`
- `oaTaper`
- `oaInputTaper`
- `oaOutputTaper`

- `oaShielding`
- `oaTransReflexive`
- `oaReflexive`
- `oaInterChild`
- `oaUserDefined`

Support for Predefined Via Parameters

OpenAccess lets you store predefined parameterizations of oaStdVias and oaCustomVias. These predefined parameterizations of the via constraints are known as via variants. Thus, the oaViaVariant object represents a named pairing of an oaViaDef reference and a specified set of parameters. oaViaVariants can be stored in technology and design databases.

The ASCII technology file reader and writer support the oaViaVariants to preserve technology data integrity. Though Virtuoso supports these objects, it does not create them automatically.

Purpose-Aware Constraints

OpenAccess data model 4 supports purpose-aware constraints. This allows applications to limit the scope of a layer-based constraint to a specific set of purposes. These constraints are aimed for the fillOPC rules.

Cadence recommends that unless you have a good reason to define a purpose-aware constraint using the 'drawing' purpose, do not use this purpose. You cannot revert back once the database has been DM4 stamped.

Reserved Purposes

In data model 4, OpenAccess has introduced built-in purpose types for matching a layer with a constraint. The `oaFillopcPurposeType` built-in purpose type specifies whether or not a fill shape requires OPC.

Support for New ITDB Constructs

processFamily

Foundries can set the this technology database attribute to ensure that technology databases in the same graph are from the same process family.

exclusiveLayers (layer attribute):

OpenAccess allows you to exclude layers by name from a graph of technology databases. This helps you prevent inadvertent references to incompatible layers in a graph of technology databases.

From IC6.1.2, you can create such constructs by explicitly adding them to your ASCII technology file and compiling it. These constructs are not mandatory. Virtuoso does not create these constructs automatically.

Constraint Parameters

There are new built-in `oaConstraintParamTypes` for which applications can create corresponding `oaConstraintParams`.

- `oacSpacingDirectionParamType`
- `oacCutDistanceConstraintParamType`
- `oacDistanceWithinConstraintParamType`
- `oacNoSharedEdgeConstraintParamType`
- `oacNotchLengthConstraintParamType`
- `oacNotchSpacingConstraintParamType`
- `oacNotchWidthConstraintParamType`
- `oacConnectivityTypeConstraintParamType`
- `oacPGNetConstraintParamType`
- `oacCenterToCenterConstraintParamType`
- `oacAreaConstraintParamType`
- `oacStackConstraintParamType`

- `oacExceptSamePGNetConstraintParamType`
- `oacNoSingleCutViaConstraintParamType`

The following constraint parameter has been modified:

- `oacWidthLengthTableTypeConstraintParamType`

  This parameter is extended to accept a third integer value to designate that the constraint value stores the LEF `SPACINGTABLE PARALLELRUNLENGTH TWOWIDTHS` specification.

Huge String/Name Data

OpenAccess provides the ability to open very large databases containing over 4 GB of string data or over 2GB of string data from object names represented using `oaName`.

oaPartitions

OpenAccess provides the ability to access only the data that an application needs, improving both capacity and performance. Specifically, the `oaPartition` class lets an application organize data of a given data type into different partitions, which can then be loaded individually.

Huge oaPointArray Data

OpenAccess supports the ability to create more than 4GB of `pointArray` data on any one of the following design database objects: polygons, lines, paths, blockages, boundaries, and markers.

Huge oaAppProp Data

OpenAccess supports the ability to create more than 4GB of `AppProp` data in any OpenAccess database.

Moving Your Data to OpenAccess Data Model 4

When you open a database in IC6.1.2, it gets stamped as data model 4. For example, the use of the following constraints or features in IC6.1.2 triggers a data model 4 uprev on your data.

When you save the design or technology database using these constraints or features, the data model 4 stamp will be attached to your database.

Once your database is stamped with any data model stamp, you will not be able to open it in an earlier release.

- When a technology database that contains purpose-specific rules is loaded, the database is uprev'ed to data model 4.

  For example,

```
minSpacing ( metal1 0.4)                => will not trigger DM4
minSpacing ( ('metal1' 'fillOPC') 0.5) => will trigger DM4
minSpacing ( ('metal1' 'drawing') 0.4) => will trigger DM4
```

- Any technology database referring to the `oaFillopcPurposeType` purpose in an LPP-based rule or any design database containing shapes on this purpose are uprev'ed to data model 4 when loaded; the DFII reserved purpose number is uprev'ed to the OpenAccess system reserved purpose.

  If a database created before data model 4 includes a user-defined purpose that uses a purpose number in the reserved range, and that database is loaded by newer OpenAccess shared libraries, the user-defined purposes in the reserved range are removed from the database.

- Use of the `'select` operator in a derived layer construct of the ASCII technology file results in the corresponding technology database to be a data model 4 technology database.

  For example,

```
techDerivedLayers(
  ; use purpose name allowed
    ( drv3   10003   (metal2  'select  drawing) )
  ; use purpose name allowed
    ( drv4   10004   (via     'select  label) )
  ; purpose number allowed
    ( drv5   10003   (metal2  'select  10) )
; techDerivedLayers
```

- Setting the following attributes to anything different than the default value triggers data model 4 stamping for the design database.

```
pinId ~> type
layerBlockageID ~> allowPGNet
layerBlockageID ~> spacing
areaBlockageID ~> isSoft
```

```
areaHaloID ~> isSoft
```

Opening an IC6.1.1 Database

Whether or not a data model 4 feature is added to your database in IC6.1.2 and you attempt to open the technology or design database in IC6.1.1, you will receive one of the following messages based on the action you are trying to perform:

- If you open a technology database that contains "do not open" DM4 features, you get the following warning message:

  ```
  *WARNING* techGetObjTechFile: Tech database contains features defined in
  OpenAcess data model version 4, and cannot be opened by software that supports
  data model version 3
  ```

- If you open a design database that contains "do not open" DM4 features, you get the following warning message:

  ```
  *WARNING* dbOpenCellViewByType: Design database contains features defined in
  OpenAcess data model version 4, and cannot be opened by software that supports
  data model version 3
  ```

- If you open a design database that contains "do not append" DM4 features, you get the following warning message:

  ```
  *WARNING* dbOpenCellViewByType: Design database contains features defined in
  OpenAcess data model version 4, and cannot be opened for edit by software that
  only supports data model version 3. It can be opened for read.
  ```

Virtuoso Support of Technology File Constraints

Technology File Updates Using Merge and Replace on page 106

Changes to Table Spacing Syntax on page 107

Translation of the resistancePerCut Property on page 108

In DFII on CDB, the technology file rules were implemented as hard foundry technology rules. In the Virtuoso Studio design environment, technology rules are replaced with technology constraints which are grouped into constraint groups.

Each constraint group contains subsections which are used to divide similar types of constraints into smaller sections. These sections closely correspond with the CDB rule sections. For example, the following CDB technology file class is mapped to the Virtuoso environment technology file as follows;

CDB Technology File	Virtuoso Studio design environment technology file

`physicalRules(`
`spacingRules()`
	`constraintGroups(`
	`("foundry"`
	`spacings()`

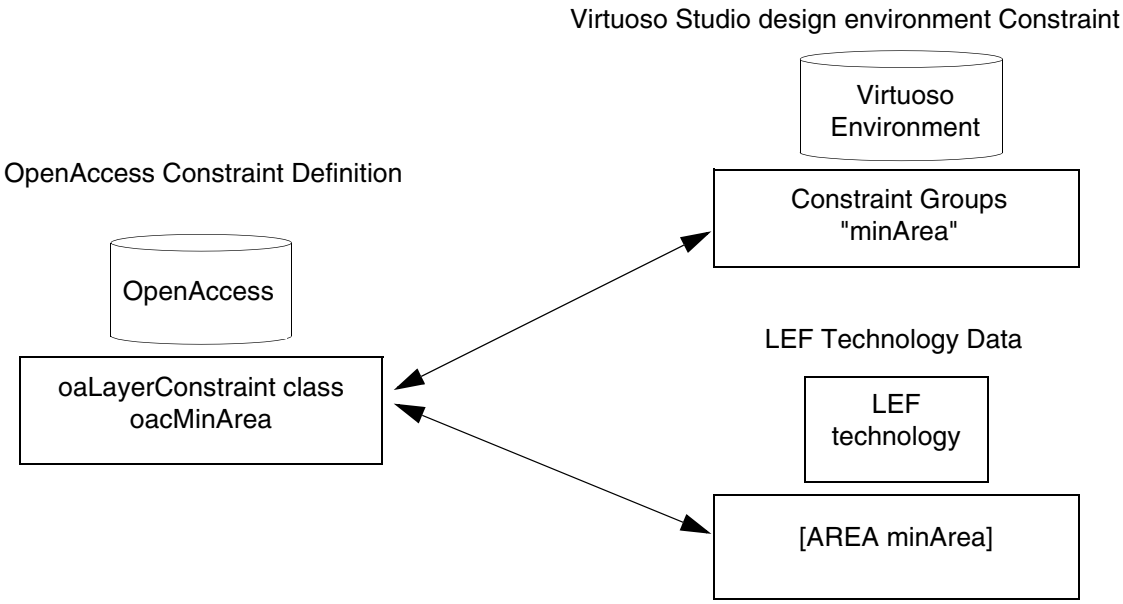
For information about types of constraint groups, see Types of Constraint Groups on page 101.

Constraints and constraint groups (containers that hold constraints and references to constraint groups) can be defined in the technology file or are created when technology data is translated from an external application. Constraints that have been created by an external application (with proper name and definition) and translated into the Virtuoso environment are mapped to an appropriate constraint and constraint group in the technology file.

Whether or not a constraint group is applied to an object, depends on the application. Constraints are obeyed only in the event an application is designed to adhere to them. The Virtuoso application support is based on the applicable constraint group (and thus the contained constraints) and of the constraints in that group, which are designated to be enforced or obeyed by the application.

In the diagram below, `minArea` is defined as single layer spacing constraint in the Virtuoso technology file and is mapped to the constraint type `oacMinArea`. The LEF technology data

{`AREA minArea`} also maps to the constraint type `oacMinArea`. Using the predefined name and explicit definition allows interoperability between tools.



User Defined Constraints

CDB rules translated to the Virtuoso environment can be added to two types of sections within a constraint group.

- CDB rules that map correctly to the Virtuoso environment constraint definitions and are interoperable with other applications on OpenAccess.
- User defined constraints which are CDB rules that have a unique name and can not be mapped to an OpenAccess constraint. User defined constraints are only accessible to Virtuoso applications and are not interoperable with other tools. This includes layer purpose pair (LPP) based constraints. Technology file constraints that are defined for LPPs are stored as a private extensions and are only available for Virtuoso applications.

  For information about how rules are mapped, see Mapping CDB Physical Technology File Rules to OpenAccess Constraints on page 156.

OpenAccess constraints support an implicit order of precedence. User defined constraints do not support an order of precedence. Because of this, user defined constraints are grouped below all OpenAccess constraints that do require order.

For example, a CDB technology file containing `spacingRules`, `orderedSpacingRules`, and `tableSpacingRules`, when translated and dumped, could contain six sections,

- user defined `spacingRules`
- OpenAccess `spacingRules`
- user defined `orderedSpacingRules`
- OpenAccess `orderedSpacingRules`
- user defined `tableSpacingRules`
- OpenAccess `tableSpacingRules`

The user defined spacing rule, `minMosWidth`, is defined as:

```
constraintGroups(
( "foundry"
;physical constraints
   spacings(
   ( minSpacing tx_layer g_value )
   ) ;spacings
   spacings(
   ( minMosWidth tx_layer g_value )
   ) ;spacings
) ;foundry
) ;constraintGroups
```

The value of a user defined constraint is applied as a string. The interpretation of the string is handled by individual applications.

Virtuoso Studio design environment ASCII Technology File Definition:

```
t_userConstraint lt_layer1 [lt_layer2] t_value )
```

Example:

```
(userConstraint metal1 "myValue")
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Value type: `oaStringValue`

Types of Constraint Groups

There are different types of constraint groups which are supported: foundry and setup constraint groups. Constraint groups are applied to the design associated with the attached technology library.

Foundry Constraint Group

The `foundry` constraint group consists of the set of technology constraints that must always be satisfied. These constraints are implicitly applied to all designs that are associated with a technology database containing this set of constraints. For example:

```
constraintGroups(
   ( "foundry"
```

The name of the hard foundry constraint group defined in the ASCII technology file is `foundry`. The name `foundry` is only defined in the Virtuoso environment. The OpenAccess foundry constraint group is not associated with a specific name.

Setup Constraint Group

Setup constraint groups are used by the XL layout editor and the chip assembly router. The default names of the setup constraint groups are:

- `virtuosoDefaultExtractorSetup`: XL layout editor
- `virtuosoDefaultSetup`: chip assembly router.

The `virtuosoDefaultExtractorSetup` constraint group lists the valid layers to be used for XL layout editor connectivity extraction and the layers to be used by the *Create Wire* command within the XL layout editor.

Setup constraint groups can be created using any name and can be selected from the options form of the corresponding application.

Specialized Constraint Groups

The following user-defined constraint groups are created by OpenAccess LEF and DEF translators:

- `LEFDefaultRouteSpec`: Stores routing information, such as information about layers and vias
- `LEFSpecialRouteSpec`: Stores information about power routing vias

The specialized constraint groups can be stored in either technology or design database. If stored in a technology database, the constraints apply across any design that is associated with that technology. If stored in the design database, the constraints apply only to that particular design. Design-specific constraints take precedence over constraints in the technology database.

A single technology can have only one `LEFDefaultRouteSpec` and one `LEFSpecialRouteSpec`. However, each technology in a graph of technology databases can have its own set of these constraint groups. In addition, a design can have its own `LEFDefaultRouteSpec` and `LEFSpecialRouteSpec` constraint group. Therefore, an application can have multiple LEFDefaultRouteSpecs and multiple LEFSpecialRouteSpecs in a graph of referenced technology databases. An application creates overriding constraints in the more derived technology.

Older databases do not allow multiple constraint groups named `LEFDefaultRouteSpec` because they conflict by name. In OpenAccess data model 4, by using the constraint group definition, a graph can have multiple constraint groups of these types as long as the constraint groups names are unique in the graph.

Note: A design-specific `LEFDefaultRouteSpec` or `LEFSpecialRouteSpec` need not be complete; rather, it can contain a subset of constraints as overrides to the constraints specified in the technology database.

For information about how OpenAccess LEF and DEF translators map constructs into LEFDefaultRouteSpec and LEFSpecialRouteSpec, see Design Data and Technology Data Preparation in the *Mixed Signal (MS) Interoperability Guide*.

Member Constraint Group

A constraint group can be created as a member of another constraint group. When multiple member constraint groups are specified, their order of precedence is from top to bottom; that is, the first member constraint group specified has the highest precedence, and the last has the lowest.

Built-In Constraint Group Types

Most of the built-in constraint group types are for the design database only. However, some of them can be specified in the technology database as well.

Virtuoso Studio design environment ASCII Technology File Definition:

```
constraintGroups(
; name [override] [type]
  ( "group" [t | nil] [nil | "taper" | "inputTaper" | "outputTaper"]
  ...
  ) ; group
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaTaper`, `oaInputTaper`, `oaOutputTaper`
Constraint type: `oaConstraintGroupTypeEnum`

LEF Syntax:

NA

Hard and Soft Constraints

In Virtuoso, soft constraints saved to Virtuoso or translated to Virtuoso are preserved in the technology file. However, Virtuoso applications do not support soft constraints. The hard and soft indicators are as follows:

```
( t_constraint [lt_layers] g_value 'hard )
( t_constraint [lt_layers] g_value 'soft )
```

Coincident Allowed Parameter

The coincident allowed parameter, which can be specified for several constraints, is defined using the `'coincidentAllowed` symbol. It cannot be specified for user-defined or layer-purpose pair constraints.

```
( t_constraint [lt_layers] g_value 'coincidentAllowed 'hard)
( t_constraint [lt_layers] g_value 'soft 'coincidentAllowed)
```

Note: The `'coincidentAllowed` symbol applies only to OpenAccess 2.2 constraints, not LPPs or user constraints. For more information, see Coincident Extension Allowed Constraint.

Constraint Group Properties

In the Virtuoso environment, properties can be associated with a constraint group. The following is an example of a property applied to a constraint group:

```
constraintGroups(
;( group
( "constraintGroupName"
...
   properties(
   ( t_propName g_value )
   ); properties
) ;constraintGroupName
); constraintGroups
```

Constraint Group Semantics

Constraint groups follow the precedence semantic as per which, the first hard constraint found in the constraint group must be satisfied. If a soft constraint is encountered and not satisfied, the search for constraint continues until either a constraint that can be satisfied is found or a hard constraint that cannot be satisfied is found.

To handle the more complex rules of 45/65 nm processes where constraints are specified as either a series of circumstances in which a given constraint applies or a set of exception conditions in which the constraint does not apply, the following constraint group semantics are used:

- AND: This operator specifies that all constraints within the same constraint group must be satisfied.
- OR: This operator specifies that only one of constraints within the same constraint group must be satisfied.

The precedence semantic is the default semantic followed by constraint groups.

Virtuoso Studio design environment ASCII Technology File Definition:

```
constraintGroups(
; name [override] [type] [semantic]
  ( "group" [t | nil] [nil] ['and | 'or | 'precedence]
  ...
  ) ; group
) ;constraintGroups
```

OpenAccess Constraint:

Semantic-type attribute on constraint group: `enum`

For example, the following LEF representation of minimum protrusion number of cuts constraint for layer VIA 1:

```
MINIMUMCUT 2 WIDTH 0.419 LENGTH 0.419 WITHIN 5.000;
MINIMUMCUT 4 WIDTH 0.979 LENGTH 0.979 WITHIN 6.000;
```

would map to the following constraint group using the AND operator:

```
constraintGroups(
  ( "myAndConstraintGroup" nil nil 'and
     spacings(
     ( minProtrusionNumCut VIA1 'distance 5.000 'width 0.419 'length 0.419 2 )
     ( minProtrusionNumCut VIA1 'distance 6.000 'width 0.979 'length 0.979 4 )
     ) ;spacings
  ) ;myAndConstraintGroup
) ;constraintGroups
```

Note: You cannot define other constraint groups as members in constraint groups in which the AND or OR operators are used.

Technology Database File Name

The name of the technology database file has changed from `techfile.cds` to `tech.db`; however, do not hard code the filename. Applications that use a hard-coded technology filename should no longer do so. Instead, if you require the technology filename, retrieve it as follows:

- For SKILL, use the new SKILL function `techGetDefaultTechName`
- For C and C++, use the `# define techcDefaultTechFileName` constant

Technology File Updates Using Merge and Replace

The following describes the behavior of loading a technology library with an ASCII technology file using the *Merge* and *Replace* options on the *Load Technology File* form.

- Using Merge does not manipulate constraint groups.

  When updating a technology library using the *Merge* option, existing constraint groups that are not referenced in ASCII version of the technology file are not manipulated.

- Using *Replace* removes constraint groups.

  When updating a technology library using the *Replace* option, existing constraint groups that are not referenced in ASCII version of the technology file are removed.

For example, if the existing technology file contains constraint groups A, B, and C, and the ASCII version contains A' and D (A' being a modified constraint group) the results of updating would be:

- Using the *Merge* option would result in a technology library containing constraint groups A', B, C and D.
- Using the *Replace* option would result in a technology library containing constraint groups A' and D.

Note: In both cases, constraint group A is completely replaced by A'. For example, if you update an existing technology library which contains constraint group A (specifying constraints `minArea`, `minWidth`, and `minNumCut`) and the modified ASCII contains a constraint group A (specifying the constraint `minExtension`) the updated constraint group A will only contain the constraint `minExtension`.

When updating the technology library using both Merge and Replace option,

- new constraint groups referenced in ASCII version of the technology file are added.
- same name constraint groups in the existing technology file are replaced with the ASCII version of the technology file.

Changes to Table Spacing Syntax

Virtuoso Environment Syntax

In the `spacingTables()` syntax, each table index definition is given by:

`l_index1Definitions` is a list defining the name, predefined index values, and user defined match function for the first table dimension. The format of the list is as follows:

```
( [nt_indexName] [(lg_indexValue...)] [t_userMatchFunction] )
```

- where `nt_indexName` 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.
- The `g_indexValue` is a list of predefined indexes, specifying the indexes that can be used in the table. Valid values: Any number or string.
- The `t_userMatchFunction` is the name of a user-defined match function. Valid values: Any string.

CDB Syntax

In CDB, entries in the table are defined in the following way:

the `l_table` is a list defining the table entries, in the following format:

```
( g_index1 [t_matchType1] [ g_index2 [t_matchType2] ] g_value ... )
```

- where `g_index1` is the first index in a two dimensional table or the only index in a one dimensional table. Valid values: Any number, string, or list.
- The `t_matchType1` is the match type to use for index1. Valid values: One of the following: `< , <=, ==, >=, >`, `userMatchFunction` or `nil` where `userMatchFunction` is a user-defined match function. Default: `==`.
- The `g_index2` is the second index in a two dimensional table. Valid values: Any number, string, or list.

- `t_matchType2` is the match type to use for `index2`. Valid values: One of the following: `<, <=, ==, >=, >, t_userMatchFunction` or `nil` where `userMatchFunction` is a user-defined match function Default: `==`.
- `g_value` is the value to use when the application finds a match to the index or pair of indexes during table lookup. Valid values: Any number or string.

In the Virtuoso environment technology file of any table, a unified "match function" is defined, either `">=" (">") or "<=" (<)`, depending on the semantic defined for the constraint the table is representing, except for on the index boundaries (lowest or highest index value) which you may specify differently. The indexes of the table are sorted in ascending order.

Currently, the match function, if it is defined, is ignored. All tables are predefined with "`>=`" look up. When the technology file is translated from CDB to Virtuoso environment, the translation does not convert the match function since this is not supported. During translation, the table is populated by the correct values from the CDB table taking into consideration any match functions which may be defined.

Translation of the resistancePerCut Property

If the property `resistancePerCut` exists on cellview, the value of the property is translated to `resistancePerCut` of the via definition.

If the property `res` exists on cellview, the value is translated to `resistancePerCut` of the via definition as the value of property multiplied by the number of cuts.

If the electrical rule `resistancePerCut` exists on the via layer, the value is translated to `resistancePerCut` of the via definition.

Conversion of Technology File controls Class

CDB	Virtuoso Studio design environment
controls techParams iccRules techPermissions	controls techParams techPermissions viewTypeUnits mfgGridResolutionZ refTechLibs

New:

viewTypeUnits: DBUPerUU Stored in the Technology File on page 112

mfgGridResolution:Manufacturing Grid on page 113

Layers

Virtuoso Studio design environment ASCII Technology File Definition:

```
constraintGroups(
( "virtuosoDefaultSetup"
   interconnect(
   (validLayers (tx_layer)... )
   ) ;interconnect
) ;virtuosoDefaultSetup
) ;constraintGroups

layerRules(
   functions(
      ( tx_layer tx_material g_maskNumber)
   ); functions
);layerRules
```

Layer purposes in `iccLayers()` are ignored.

DFII on CDB ASCII Technology File Definition:

```
iccLayers = list(
   list(lt_layer t_function t_direction f_width f_spacing [g_ref
[g_translate]])
   [list(...)]...
)
```

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacValidRoutingLayers`
Value type: `oaLayerArrayValue`

Class: `oaConstraintGroup`, `virtuosoDefaultSetup`()

Vias

Virtuoso Studio design environment ASCII Technology File Definition:

```
constraintGroups(
( "virtuosoDefaultSetup"
   interconnect(
   (validVias (tx_layer)... )
   ) ;interconnect
) ;virtuosoDefaultSetup
) ;constraintGroups
```

For cdsViaDevices: `standardViaDefs()`

```
viaDefs(
standardViaDefs(
   ( t_viaDefName tx_layer1 tx_layer2
      ( t_cutLayer n_cutWidth n_cutHeight [n_resistancePerCut] )
      ( x_cutRows x_cutCol ( l_cutSpace ) )
       l_layer1Enc ) ( l_layer2Enc ) ( l_layer1Offset )
      ( l_layer2Offset ) ( l_origOffset )
      [tx_implant1 (l_implant1Enc) [tx_implant2 (l_implant2Enc]]
   )
) ;standardViaDefs
) ;viaDefs
```

All other vias: `customViaDefs()`

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

For more information about mapping CDB vias, see Mapping of CDB Devices.

DFII on CDB ASCII Technology File Definition:

```
iccVias= list(
   list(
       list([list(t_libName t_cellName t_viewName)] |
       t_pseudoViaName list(lt_layer ....)
       [g_translate])
       [list(...)]...
)
```

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacValidRoutingVias`
Value type: `oaLayerArrayValue`

Class: `oaConstraintGroup`, `virtuosoDefaultSetup`()

DBUPerUU Stored in the Technology File

Unlike CDB, OpenAccess provides a single location for database units per user unit (`DBUPerUU`) and user unit information (`userUnit`) stored in the technology file. The following is an example of the ASCII technology file syntax.

```
controls(
   viewTypeUnits(
   ( maskLayout          "micron"        2000            )
   ( schematic           "inch"          160             )
   ( schematicSymbol     "inch"          160             )
   ( netlist             "inch"          160             )
   )
)
```

When migrating data from CDB to the Virtuoso environment, the values for user units (userUnits) and DBUPerUU are obtained from various locations and consolidated at a single location in the technology file.

The value of `0.0` is accepted in the technology file. If the value of `0.0` is unintentionally set in the technology file, the bbox of a new cellview and the initial window zoom level would be infinite. Also, the technology file API (see below) would return a value of (NAN:NAN:NAN:NAN).

DBUPerUU Access

The APIs listed below are provided to access `DBUPerUU` and `userUnit` information in order to meet the new model. These new APIs are implemented only for OpenAccess-based technology file APIs.

`techGetDBUPerUU(`*`d_techFileId t_cvType`*`)`

`techSetDBUPerUU(`*`d_techFileId t_cvType f_dbuperuu`*`)`

`techGetUserUnit(`*`d_techFileId t_cvType`*`)`

`techSetUserUnit(`*`d_techFileId t_cvType t_userUnit`*`)`

Using `lib~>DBUPerUU` or `lib~>userUnits` will report a warning. Also, `lib~>prop` will not return `DBUPerUU` and `userUnits` as properties. Any SKILL procedures depending on these calls should be modified.

For example, if the original SKILL code set the `dbu/uu` as

```
lib~>DBUPerUU~>maskLayout = 2000.0
```

the code based on OpenAccess should be

```
techSetDBUPerUU(techGetTechFile(lib) "maskLayout" 2000.0)
```

Manufacturing Grid

In CDB, the manufacturing grid was stored in the `physicalRules` class. In Virtuoso, the manufacturing grid is stored in the `controls` section of the technology file.

Manufacturing Resolution

Virtuoso Studio design environment ASCII Technology File Definition:

```
controls(
   mfgGridResolution( g_resolution )
) ;controls
```

DFII on CDB ASCII Technology File Definition:

```
physicalRules(
   mfgGridResolution( g_resolution )
) ;physicalRules
```

OpenAccess Constraint:

Class: `oaTech`
`::getDefaultManufacturingGrid()`
Class: `oaPhysicalLayer`
`::resetManufacturingGrid()`

Conversion of Technology File layerDefinitions Class

CDB	Virtuoso Studio design environment
layerDefinitions techLayers techPurposes techLayerPurposePriorities techDisplays techLayerProperties	layerDefinitions techPurposes techLayers techDerivedLayers techLayerPurposePriorities techDisplays techLayerProperties

Changes:

techPurposes:

Technology File Changes for Layer Purposes on page 115

LEF In Changes for Valid Layer Purposes on page 117

OpenAccess Constants for Reserved Purposes on page 118

Different Purpose and Layer Numbers Values on page 120

For C and C++ Code, Use Cadence-Defined Constants and typedefs on page 121

Location of Cadence-defined Constants and typedefs on page 124

techLayers:

Technology File Changes for Layers on page 125

Virtuoso Environment Layer Purpose Pairs Support on page 126

Technology File and LPP SKILL Attribute Changes on page 126

techDerivedLayers:

Derived Layers

Sized Layers

techLayerProperties: Characterization Rules Stored as Layer Properties

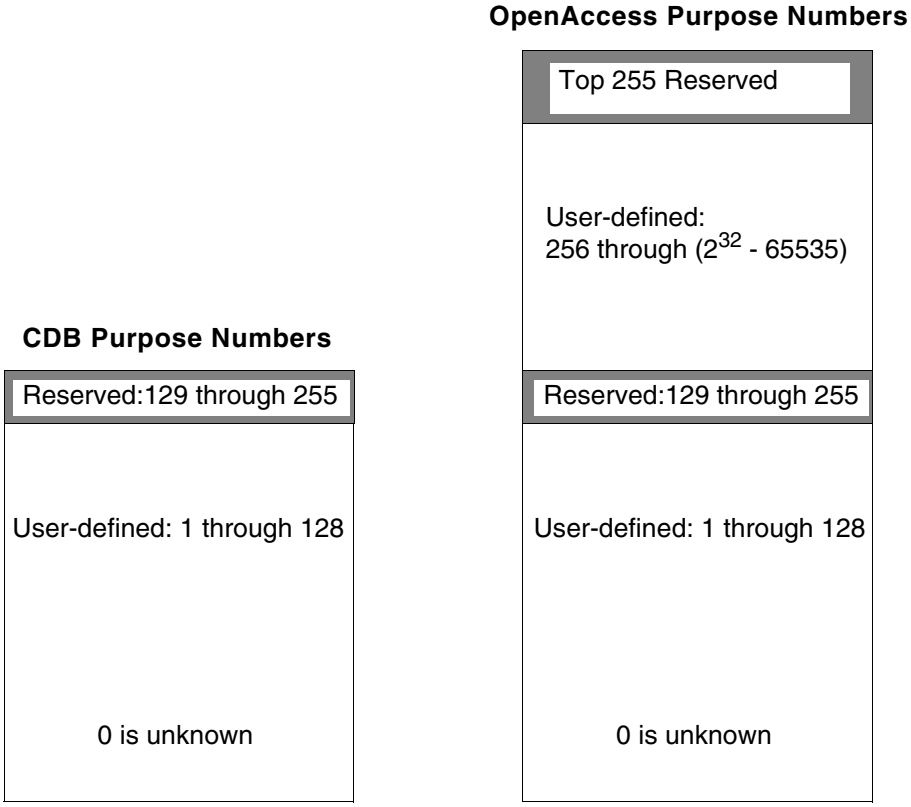
Technology File Changes for Layer Purposes

The purpose 0 is “unknown” (reserved).

For OpenAccess, the number of purposes has been increased as follows:

- You can define purposes numbered 1 through 128.
- If you want to define more purposes, you need to define purposes between 256 through 2^{32} - 65535.

  If you need to define more than 128 purposes, start with purpose number 256 and increment it by one to avoid conflict with purpose numbers at the high end of the range that are reserved by Cadence for future system use.



Virtuoso Studio design environment Technology File Reserved Purpose Types

The following table lists the Virtuoso reserved purposes that support control of display and selection of specific objects. The LPPs for the objects must be present in the technology file

and the corresponding packets must be present in the *display.drf* file. Highlighted system reserved purposes are new in the Virtuoso Studio design environment.

System Reserve Purpose	Purpose #	Abbreviation
redundant	-8	
customFill	-12	
gapFill	-7	
annotation	-6	
OPCAntiSerif	-5	
OPCSerif	-4	
slot	-3	
fill	-2	fil
fillOPC	-11	fio
drawing	-1	drw
fatal	223	fat
critical	224	crt
soCritical	225	scr
soError	226	ser
ackWarn	227	ack
info	228	inf
track	229	trk
blockage	230	blk
grid	231	grd
warning	234	wng
tool1	235	tl1
tool0	236	tl0
label	237	lbl
flight	238	flt
error	239	err

System Reserve Purpose	Purpose #	Abbreviation
annotate	240	ant
drawing1	241	dr1
drawing2	242	dr2
drawing3	243	dr3
drawing4	244	dr4
drawing5	245	dr5
drawing6	246	dr6
drawing7	247	dr7
drawing8	248	dr8
drawing9	249	dr9
boundary	250	bnd
pin	251	pin
net	253	net
cell	254	cel
all	255	all

Note: No purpose 252 exists on OpenAccess. If you attempt to use the purpose 252 using SKILL in OpenAccess environment, it gets mapped to the `drawing` purpose (-1) on OpenAccess.

Changing the Number of Characters for LSW Purpose Abbreviation

To change the display from 3 to 2 characters for the purpose abbreviation in the LSW, use the layout environment variable `lswLPIconPurposeLength`. For example,

```
envSetVal("layout" "lswLPIconPurposeLength" 'int 2)
```

LEF In Changes for Valid Layer Purposes

When translating data into the Virtuoso environment using LEF In, the following layer purposes are by default not valid, meaning they do not appear in the LSW. These purposes are used for display of objects only and the LPPs will not be available for shape drawing from

the *Layer* panel of the LSW. Control selection and display of objects through the *Object* panel of the LSW.

```
net
pin
slot
fill
gapFill
blockage
boundary
```

The `lefin` option -`pinPurp` allows you to specify a specific purpose for pins, such as "`pin`", when translating data. By default, pins are created on the purpose "`drawing`". You can specify any purpose name. If the purpose does not exist in the technology library then `lefin` will create the purpose and add the name to the technology file. The purpose chosen as the pin purpose will be valid by default. This change only effects technology files newly created by LEF In.

OpenAccess Constants for Reserved Purposes

The following table shows the Cadence-defined constants for reserved purposes for the OpenAccess database. These predefined purposes are always available, and may not be changed nor removed. The *Constant for Purpose Name* always has the data type of string, while the *Constant for Purpose Number* has its data type determined by the typedef, `dbPurpose`.

Purpose Name **Constant for Purpose Name** **Constant for Purpose Number** **Value**	**Description**

`redundant`
`oacPurposeRedundant`
`oavPurposeNumberRedundant`
$2^{32} - 8$
	The redundant purpose is used to flag specific objects that have been placed more than once, such as double vias on a route.

`gapFill`
`oacPurposeGapFill`
`oavPurposeNumberGapFill`
$2^{32} - 7$
	Elements marked with the gapFill purpose are used to fill large open areas of a mask to increase planarization of the layer during manufacturing. These elements are not part of any electrical network on the chip.

Purpose Name **Constant for Purpose Name** **Constant for Purpose Number** **Value**	**Description**
annotation oacPurposeAnnotation oavPurposeNumberOPCAntiSerif 2^{32} - 6	Used for chip annotation, such as labels and logos
OPCAntiSerif oacPurposeOPCAntiSerif oavPurposeNumberOPCAntiSerif 2^{32} - 5	Identifies data created for optical proximity correction, in which data is removed from drawn masks.
OPCSerif oacPurposeOPCSerif oavPurposeNumberOPCSerif 2^{32} - 4	Identifies data created by optical proximity correction, in which data is added to drawn masks.
slot oacPurposeSlot oavPurposeNumberSlot 2^{32} - 3	Identifies shapes to process for slotting modification to avoid errors during the manufacturing process.
fill oacPurposeFill oavPurposeNumberDrawing 2^{32} - 2	Identifies shapes as fill shapes to satisfy the requirement that a certain percentage of your chip be made from a specific material.
drawing oacPurposeDrawing oavPurposeNumberDrawing 2^{32} - 1	Default purpose used for the creation of objects.

Different Purpose and Layer Numbers Values

In CDB, the `drawing` purpose is represented by the number 252. In OpenAccess the `drawing` purpose is represented by 2^{32} - 1, which in hexadecimal is the number `0xffffffff`.

The range of values for both purpose and layer numbers is much higher for OpenAccess than for CDB, due to the larger number of bits available. And for OpenAccess, the values of some reserved purpose numbers are assigned differently than for CDB.

- CDB uses 8-bit unsigned characters to represent positive numbers, in the range 0 through 255, which can also be stated as 0 through 2^8 - 1.
- OpenAccess uses 32-bit unsigned integers to represent positive numbers, in the range of 0 through 2^{32} - 1. Large numbers in this range are represented in hexadecimal.

Purpose and Layers Numbers Can Appear as Negative Numbers.

SKILL can handle only signed numbers; therefore, in OpenAccess, large, unsigned numbers appear as negative numbers in SKILL, and when printed out by SKILL to ASCII.

For example, in OpenAccess, the drawing purpose is represented by 2^{32} - 1, which in hexadecimal is the number `0xffffffff`.

In C or C++, the unsigned number `0xffffffff` is available “as is”, however, in SKILL and ASCII produced by SKILL, it is not. In SKILL or ASCII, the number `0xffffffff` is treated as signed and appears as -1, a negative number.

Updating User-Defined Purpose Names

Important

If you defined a purpose, such as `fill`, to mean something other than the OpenAccess definition, you must change the name of the purpose in your technology file to another purpose name, such as `myFill`, `mySlot`, `myAnnotation`, and so forth. You must change the name in CDB and then translate the data to OpenAccess.

Updating User Defined Purposes and Hard Coded Values

You will need to update your code if your code contains hard-coded purpose number such as 252 to represent the `drawing` purpose. If your code contains user defined purposes that

conflict with OpenAccess purposes or CDB-specific hard-coded purpose numbers, your code will not work correctly with the OpenAccess database until you replace them.

Ideally, you should replace hard-coded numbers with their Cadence-defined names; for example, replace the number 252 with the string name, `drawing`. It is less desirable to replace CDB-specific hard-coded numbers with OpenAccess-specific hard-coded numbers, since hard-coded numbers are likely to require more maintenance in the future.

However, if you must use the purpose number instead of the purpose name in OpenAccess, you will have to use the negative number, which in SKILL, is equivalent to the large, unsigned number in OpenAccess. For example, you would have to use -1 for the `drawing` purpose.

In C and C++ code replace user defined purposes that conflict with OpenAccess purposes or hard coded values or with the OpenAccess-specific constant for the purpose name or purpose number. See OpenAccess Constants for Reserved Purposes on page 118.

Reload your code before you translate your data, especially if it includes SKILL-based Pcells.

Updating Data Type Definitions

You will need to update your code if your code contains CDB specific base data type definitions for purpose (such as `unsigned char`) instead of a Cadence-defined typedef (such as `techLayerNum`)

In SKILL, there are no Cadence-defined constants for purpose and layer numbers; nor are there typedefs. Typedefs are not needed, because SKILL automatically determines the data type for purpose and layer numbers, so no change is needed to SKILL code for data types

For C and C++ Code, Use Cadence-Defined Constants and typedefs

Cadence-Defined Constants

The `drawing` purpose is the only Cadence-defined purpose name that is common to both CDB and OpenAccess; however, it is defined by different purpose numbers. To make your C and C++ code database independent for the `drawing` purpose, use the following Cadence-defined constants for the `drawing` purpose:

- The database constant is `dbcDrawingPurpose`
- The technology file constant is `techDrawingPurpose`

These two constants are common to CDB and OpenAccess; using them helps your code run on both databases, and makes moving from CDB to OpenAccess easier.

Note: Cadence recommends that you always use Cadence-defined constants and typedefs instead of the underlying numbers and base data types.

In addition to `drawing`, CDB has numerous other Cadence-defined constants for purpose numbers, while OpenAccess has only a few. There are new Cadence-defined constants for the new OpenAccess reserved purpose names, but they are specific to the OpenAccess database, as those purposes are not defined by Cadence for CDB.

Cadence-defined constants for the values of database purpose numbers appear as hash define (`#define`) statements that look like this:

For CDB:

```
#define     dbcDrawingPurpose      252
```

For OpenAccess:

```
#define     dbcDrawingPurpose      0xffffffffu
```

For the location of the files containing the Cadence-defined constants for both CDB and OpenAccess, see "Location of Cadence-defined Constants and typedefs" on page 124.

Example of Constant for a Purpose Number

For example, for the `drawing` purpose in CDB, the purpose number is represented by the integer constant 252. But instead of setting the `drawing` purpose to a specific number for either CDB or OpenAccess, you should set it to the Cadence-defined constant for the `drawing` purpose, which is `dbcDrawingPurpose` or `techDrawingPurpose`, as shown below. Then your code will work with both databases.

Avoid hard-coded numeric value:

```
func()
    {
    myPurpose = 252;
```

Instead, use constant:

```
func()
    {
    myPurpose = dbcDrawingPurpose;
```

For information about the Cadence-defined constants for purposes reserved by OpenAccess, see "OpenAccess Constants for Reserved Purposes" on page 118.

Cadence-Defined Constants typedefs

Cadence provides predefined typedefs that let you indicate the data type for the variables you declare for purpose and layer numbers. These typedefs are common to both CDB and OpenAccess. To make your C and C++ code database independent for the data types of your variables for purposes and layers, use the following Cadence-defined typedefs:

- The database typedefs are `dbPurpose` and `dbLayer`.
- The technology file typedefs are `techPurpose` and `techLayerNum`.

The Cadence-defined typedef statements for database purpose and layer numbers look like this:

For CDB:

```
typedef unsigned char       dbPurpose;
typedef unsigned char       dbLayer;
```

For OpenAccess:

```
typedef unsigned int        dbPurpose;
typedef unsigned int        dbLayer;
```

And for technology file purpose and layer numbers, they look like this:

For CDB:

```
typedef unsigned char       techPurpose;
typedef unsigned char       techLayerNum;
```

For OpenAccess:

```
typedef unsigned int        techPurpose;
typedef unsigned int        techLayerNum;
```

For the location of the files containing the Cadence-defined typedefs for variables for both CDB and OpenAccess, see "Location of Cadence-defined Constants and typedefs" on page 124.

In CDB, the data type for purpose number is unsigned char. But instead of using the base data type to declare variables, you should use the Cadence-defined typedefs; then your code will work with both CDB and OpenAccess.

Example of typedef for Purpose Numbers

For example, you need to declare a variable to contain the value of a purpose, such as `drawing`. The Cadence-defined typedef for purpose is `dbPurpose`. For purpose numbers, the base data type for CDB is `unsigned char`, while for OpenAccess, it is `unsigned int`. Instead of using the database-specific base data type, use `dbPurpose`, as shown below:

Avoid base data type:

For CDB:

```
func()
    {
    unsigned char   myPurpose;
```

For OpenAccess:

```
func()
    {
    unsigned int    myPurpose;
```

Instead, use typedef:

```
func()
    {
    dbPurpose    myPurpose;
```

For OpenAccess:

```
func()
    {
    dbPurpose    myPurpose;
```

You do not even need to know what the data type of your variable for purpose is; the Cadence system will determine which database is being used and assign the appropriate data type to `dbPurpose`.

Location of Cadence-defined Constants and typedefs

For both CDB and OpenAccess, you can view the Cadence-defined constants and typedefs in the following locations:

- For database purpose and layer numbers, in your local Cadence hierarchy:

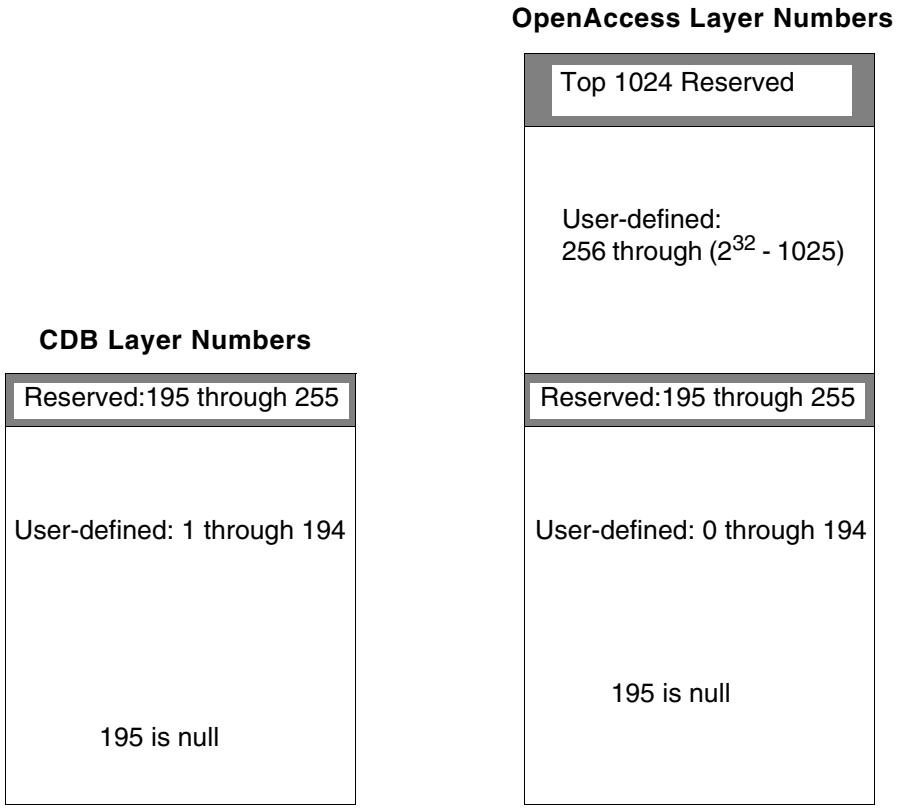
  `...tools/itkDB/include/itkDB.h`

- For technology file purpose and layer numbers, in your local Cadence hierarchy:

  `...tools/itkDB/include/itkTechdb.h`

Technology File Changes for Layers

- User-defined layers are 0 through 194 and 256 through 2^{32} - 1025.
- Cadence reserves the layers numbered 196 through 255 for system use, with layer 195 as the “null” layer and layers numbered 2^{32} - 1024 through 2^{32} - 1.



System Reserve Layers

When loading an ASCII technology file that contains user defined layers or obsolete layers within the reserved layer range, warning messages are issued and the layers are not created. For example, the following layers are obsolete as of the 4.4 release. If any of these layers still exist in your technology file you must remove them and re-load the technology file.

```
( winActiveBanner        240     winActi     )
( winAttentionText       241     winAtte     )
( winBackground          242     winBack     )
( winBorder              243     winBord     )
( winBottomShadow        244     winBott     )
( winButton              245     winButt     )
( winError               246     winErro     )
( winForeground          247     winFore     )
( winInactiveBanner      248     winInac     )
( winText                249     winText     )
```

```
( winTopShadow           250     winTopS     )
```

To remove obsolete layers or layers that are incorrectly defined in the reserved layer range, do the following.

1. In the CDB database, dump the ASCII technology file.
2. Using a text editor, find and delete incorrectly defined layers.
3. Load the ASCII technology file back into the technology library in *Replace* mode.
4. Translate the library to the Virtuoso environment.

Virtuoso Environment Layer Purpose Pairs Support

In the Virtuoso environment technology file it is possible to define technology rules for layer purpose pairs (LPP). For example, if you have a purpose named `highPower` for `metal1` then you could define a different minimum width constraint for `metal1` shapes with the `highPower` purpose from the minimum width for the `metal1` layer shapes with no purpose specified.

OpenAccess rules/constraints are only applicable to layers, not layer purpose pairs. Currently in the Virtuoso environment, any technology rule which is defined for a layer purpose pair is stored as a private extension and is available for Virtuoso applications. However, the private extension is not visible outside Virtuoso applications.

Layer purpose constraints are supported in any constraint group. Although it is not permitted to define the same layer purpose pair constraint within a constraint group.

Technology File and LPP SKILL Attribute Changes

Technology File Attributes Name Changes

The names of the following technology attributes are different for CDB and the Virtuoso environment.

Technology File Attributes	CDB return value	Virtuoso return value
`layers`	layer-purpose pairs	layers
`lps`	layer-purpose pairs	layer-purpose pairs

Technology File Attributes	CDB return value	Virtuoso return value
techlayers	layers	not applicable

The attribute `layers` only returns a list of layers in the Virtuoso environment, not LPPs. For example, to return the list of layer IDs on the layer purpose `pin`:

CDB:

```
pinLayers = setof(lp techId~>layers lp~>purpose == "pin")
```

Virtuoso Studio design environment:

```
pinLayers = setof(lp techId~>lps lp~>purpose == "pin")
```

Layer Purpose Attributes

The attribute `techLayer` is replaced by `layer` for both technology file attributes and LPP attributes.

LPP Attributes	CDB return value	Virtuoso return value
techlayers	list the layers of LPP	not applicable
layers	not applicable	list the layers of LPP

Derived Layers

Derived layer shapes are considered to be only an error if they are within a specific `oaAreaBoundary` and the `oacErrorLayer` constraint is true. The `oaAreaBoundary` is defined by the `oaConstraintGroup` to which the `oacErrorLayer` constraint belongs applied to the boundary layer object.

In the following example, the technology file contains one poly layer and two diffusion layers, `diff` and `diff2`. The combination of `diff 'and diff2` gives one oxide model and the combination `diff 'not diff2` gives a second oxide model. To identify the gate over the two oxide models, the syntax is shown below:

```
layerDefinitions(
   techDerivedLayers(
  ;( derivedLayerName layer# composition )
   ( oxide1           1000   (diff 'and diff2 ) )
   ( oxide2           1001   (diff 'not diff2 ) )
   ( gateOxide1       1002   (poly 'and oxide1 ) )
   ( gateOxide2       1003   (poly 'and oxide2 ) )
   ) ;techDerivedLayers
) ;layerDefinitions
```

The derived layer identifying `oxide1` is then used in a second derived layer, so more complex derived layers can be represented. Once the derived layer has been defined in the `layerDefinitions` section of the technology file, the error layer constraint can be defined as:

```
constraintGroups
   ( "nameOfGroup"
   ;layer constraints
   interconnect(
      ( errorLayers (tx_derivedLayer)... )
   ) ;interconnect
) ;nameOfGroup
) ;constraintGroups
```

You can specify the following types of derived layers:

- Two-Layer Derived Layers
- Purpose-Aware Derived Layers
- Sized Layers
- Area-Restricted Layers

Two-Layer Derived Layers

The logical operation symbol can be one of `'or`, `'and`, `'not`, `'xor`, `'butting`, `'buttOnly`, `'coincident`, `'coincidentOnly`, `'buttingOrCoincident`, `'overlapping`, `'buttingOrOverlapping`, `'touching`, `'inside`, `'outside`, `'avoiding`,

`'straddling`, and `'enclosing`. For information about the logical operators, see *Operator Definitions* in the Virtuoso Technology Data ASCII Files Reference.

Layer purposes are not supported in this syntax and only one logical operation can be used to represent the composition of the derived layer.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `errorLayers`

```
layerDefinitions(
   techDerivedLayers(
      ( tx_derivedLayer x_derivedLayerNum ( tx_layer1 s_operation tx_layer2)
       [x_contactCount | t_rangeVal] ['diffNet | 'sameNet] ['exclusive] )
   ) ;techDerivedLayers
) ;layerDefinitions


constraintGroups(
("foundry"
   interconnect(
      ( errorLayers (tx_derivedLayer)... )
   ) ; interconnect
) ;foundry
) ;constraintGroups
```

OpenAccess Constraint:

Class: `oaDerivedLayer` with `oaLayerOp`: `oacAndLayerOp`, `oacOrLayerOp`, `oacNotLayerOp`, `oacXorLayerOp`, `oacInsideLayerOp`, `oacOutsideLayerOp`, `oacAvoidingLayerOp`, `oacStraddlingLayerOp`, `oacButtingLayerOp`, `oacButtOnlyLayerOp`, `oacCoincidentLayerOp`, `oacCoincidentOnlyLayerOp`, `oacOverlappingLayerOp`, `oacButtingOrCoincidentLayerOp`, `oacButtingOrOverlappingLayerOp`, `oacTouchingLayerOp`, and `oacEnclosingLayerOp`
Constraint type: `oaLayerConstraint: oacErrorLayer`
Value type: `oaBooleanValue`

LEF Syntax:

NA

Purpose-Aware Derived Layers

The `'select` derived layer operation selects all shapes on the specified layer that have the required purpose. The purpose cannot be all or none and should be a specific purpose.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `errorLayers`

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

constraintGroups(
("foundry"
   interconnect(
      ( errorLayers (tx_derivedLayer)... )
   ) ; interconnect
) ;foundry
) ;constraintGroups
```

OpenAccess Constraint:

Class: `oaDerivedLayer` with `oaLayerOp`: `oacSelectLayerOp`
Constraint type: `oaLayerConstraint: oacErrorLayer`
Value type: `oaBooleanValue`
Parameter: `oaSelectShapesWithPurpose` of type `oaSelectShapesWithPurposeDerivedLayerParamType` taking an `oaPurposeValue` defining the purpose

LEF Syntax:

NA

Sized Layers

You can specify sized layers derived from sizing up or down existing layers. Although this is stored as an `oaSizedLayer` as opposed to an `oaDerivedLayer`, a sized layer can be considered to be a type of derived layer.

The operators used to derive sized layers include: `'growHorizontal`, `'growVertical`, `'shrinkHorizontal`, `'shrinkVertical`, `'grow`, and `'shrink`. The sign of the value defined indicates whether the layer should grow or shrink by the amount specified. The

horizontal and vertical directions are defined with respect to the x and y coordinates of the top-level cell.

Virtuoso Studio design environment ASCII Technology File Definition:

```
layerDefinitions(
   techDerivedLayers(
      ( tx_derivedLayer ( tx_layer s_sizeOp g_value ) )
   ) ;techDerivedLayers
) ;layerDefinitions
```

OpenAccess Constraint:

Class: `oaDerivedLayer` with `oaLayerOp`: `oacGrowLayerOp`, `oacGrowVerticalLayerOp`, `oacGrowHorizontalLayerOp`, `oacShrinkLayerOp`, `oacShrinkVerticalLayerOp`, `oacShrinkHorizontalLayerOp`,
Constraint type: `oaLayerConstraint: oacErrorLayer`
Value type: `oaBooleanValue`
Parameter: `oacDistanceDerivedLayerParamType`

Area-Restricted Layers

This constraint specifies the value to be applied to shapes on a particular layer, which are either less than or greater than the specified area.

Virtuoso Studio design environment ASCII Technology File Definition:

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

OpenAccess Constraint:

Class: `oaDerivedLayer` with `oaLayerOp`: `oacAreaLayerOp`
Parameter: `oacAreaRangeDerivedLayerParamType`

Characterization Rules Stored as Layer Properties

This section describes the set of LEF layer attributes that are not used as constraints by OpenAccess, but are stored as properties of the physical layers. These are considered to be properties of the layer itself rather than a constraint applied to the layer.

There can only be one property on a layer of a given name. When multiple properties of the same name are given, the last value found will be taken and no warning is issued.

Single Layer Area Capacitance

Virtuoso Studio design environment ASCII Technology File Definition:

Property name: `areaCapacitance`

```
layerDefinitions(
   techLayerProperties(
      (areaCapacitance tx_layer g_value )
   )
)
```

DFII on CDB ASCII Technology File Definition:

Rule name: `areaCap` and `areaCapacitance`

```
electricalRules(
   characterizationRules(
   ("areaCap" tx_layer g_value ) ... )
)
electricalRules(
   characterizationRules(
   ("areaCapacitance" tx_layer g_value ) ... )
)
```

The LEF electrical rules of the routing layer mapped to the `characterizationRules` subsection in the technology file.

OpenAccess Constraint:

Constraint type: oaLayer Property: `cLefLayerCap`

Two Layer Area Capacitance

Virtuoso Studio design environment ASCII Technology File Definition:

Property name: `areaCapacitance`

```
layerDefinitions(
   techLayerProperties(
      (areaCapacitance tx_layer1 tx_layer2 g_value )
   )
)
```

DFII on CDB ASCII Technology File Definition:

N/A

OpenAccess Constraint:

Custom Virtuoso environment two layer value.

Single Layer Edge Capacitance

Virtuoso Studio design environment ASCII Technology File Definition:

Property name: `edgeCapacitance`

```
layerDefinitions(
   techLayerProperties(
      (edgeCapacitance tx_layer g_value )
   )
)
```

DFII on CDB ASCII Technology File Definition:

Rule name: `edgeCap` and `edgeCapacitance`

```
electricalRules(
   characterizationRules(
   ("edgeCap" tx_layer g_value ) ... )
)
electricalRules(
   characterizationRules(
   ("edgeCapacitance" tx_layer g_value ) ... )
)
```

The LEF electrical rules of the routing layer mapped to the `characterizationRules` subsection in the technology file.

OpenAccess Constraint:

Constraint type: oaLayer Property: `cLefLayerEdgeCap`

Two Layer Edge Capacitance

Virtuoso Studio design environment ASCII Technology File Definition:

Property name: `edgeCapacitance` (with 2 layers specified)

```
layerDefinitions(
   techLayerProperties(
      (edgeCapacitance tx_layer1 tx_layer2 g_value )
   )
)
```

DFII on CDB ASCII Technology File Definition:

N/A

OpenAccess Constraint:

Custom Virtuoso environment two layer value.

Layer Sheet Resistance

Virtuoso Studio design environment ASCII Technology File Definition:

Property name: `sheetResistance`

```
layerDefinitions(
   techLayerProperties(
      (sheetResistance tx_layer g_value )
   )
)
```

Specifies the resistance for a square of wire in ohms per square.

DFII on CDB ASCII Technology File Definition:

Rule name: `sheetRes` and `sheetResistance`

```
electricalRules(
   characterizationRules(
   ("sheetRes" tx_layer g_value ) ... )
)
electricalRules(
   characterizationRules(
   ("sheetResistance" tx_layer g_value ) ... )
)
```

The LEF electrical rules of the routing layer mapped to the `characterizationRules` subsection in the technology file.

OpenAccess Constraint:

Constraint type: oaLayer Property: `cLefLayerRes`

Cut Layer Resistance per Cut

Virtuoso Studio design environment ASCII Technology File Definition:

Property name: `resistancePerCut`

```
layerDefinitions(
   techLayerProperties(
      (resistancePerCut tx_layer g_value )
   )
)
```

Specifies the resistance for a via in ohms per the number of cuts. This is only applicable for cut layers.

In LEF 5.6, supports a resistance per cut value for via layers. Resistance per cut applies to a layer (vias inherit resistance values if they do not have their own local values).

Also see, Translation of the resistancePerCut Property.

DFII on CDB ASCII Technology File Definition:

```
electricalRules(
   characterizationRules(
   ("sheetRes" tx_layer g_value ) ... )
)
```

When specified for cut or local interconnect layers, `sheetRes` is mapped as a property of the physical layers and mapped to `resistancePerCut`.

OpenAccess Constraint:

Constraint type: oaLayer Attribute: `cLefLayerRes`

Layer Height

Virtuoso Studio design environment ASCII Technology File Definition:

Property name: `height`

```
layerDefinitions(
   techLayerProperties(
      (height tx_layer g_value )
   )
)
```

DFII on CDB ASCII Technology File Definition:

```
electricalRules(
   characterizationRules(
   ("height" tx_layer g_value ) ... )
)
```

OpenAccess Constraint:

Constraint type: oaLayer Property: `cLefLayerHeight`

Layer Thickness

Virtuoso Studio design environment ASCII Technology File Definition:

Property name: `thickness`

```
layerDefinitions(
   techLayerProperties(
      (thickness tx_layer g_value )
   )
)
```

DFII on CDB ASCII Technology File Definition:

```
electricalRules(
   characterizationRules(
   ("thickness" tx_layer g_value ) ... )
)
```

OpenAccess Constraint:

Constraint type: oaLayer Property: `cLefLayerThickness`

Shrinkage Factor

Virtuoso Studio design environment ASCII Technology File Definition:

Property name: `shrinkage`

```
layerDefinitions(
   techLayerProperties(
      (shrinkage tx_layer g_value )
   )
)
```

DFII on CDB ASCII Technology File Definition:

```
electricalRules(
   characterizationRules(
   ("shrinkage" tx_layer g_value ) ... )
)
```

OpenAccess Constraint:

Constraint type: oaLayer Property: `cLefLayerShrinkage`

Capacitance Multiplier

Virtuoso Studio design environment ASCII Technology File Definition:

Property name: `capMultiplier`

```
layerDefinitions(
   techLayerProperties(
      (capMultiplier tx_layer g_value )
   ) ;techLayerProperties
) ;layerDefinitions
```

DFII on CDB ASCII Technology File Definition:

```
electricalRules(
   characterizationRules(
   ("capMultiplier" tx_layer g_value ) ... )
)
```

OpenAccess Constraint:

Constraint type: oaLayer Property: `cLefLayerCapMul`

Conversion of Technology File layerRules Class

CDB	Virtuoso Studio design environment
layerRules viaLayers equivalentLayers layerFunctions layerMfgResolutions layerRoutingGrids	layerRules equivalentLayers functions mfgResolutions routingDirections peakACCurrentDensity avgACCurrentDensity rmsACCurrentDensity avgDCCurrentDensity cutClasses

Note: The viaLayers class has been deprecated and will be removed in an upcoming release. Cadence recommends that you remove it from your technology files.

Changes:

viaLayers: Via Layers on page 143

equivalentLayers: Equivalent Layers on page 143

functions: Mask/Layer Functions on page 145

mfgResolutions: Manufacturing Resolution (layer) on page 147

layerRoutingGrids: Layer Routing Grids on page 147

New:

routingDirections: Preferred Routing Direction on page 148

peakACCurrentDensity: Peak AC Current Density on page 149

avgACCurrentDensity: Average AC Current Density on page 151

rmsACCurrentDensity: RMS AC Current Density on page 152

avgDCCurrentDensity: Average DC Current Density on page 153

cutClasses: Cut Classes on page 154

Via Layers

In CDB, via layers were identified in the `viaLayers` section of the `layerRules` group as follows:

```
viaLayers(
;( layer1     viaLayer      layer2   )
;( ------    --------      ------   )
( poly          cont         m1      )
( ndiff         cont         m1      )
( pdiff         cont         m1      )
( diff          cont         m1      )
( m1            via1         m2      )
( m2            via2         m3      )
( m3            via3         m4      )
( m4            via4         m5      )
) ;viaLayers
```

In the Virtuoso environment, the via layers information is determined from via definitions. See Technology File Via Definitions and Via Specifications on page 343.

If the `layerFunctions` class is not defined in CDB, mask numbers and function of layers are mapped according to the `viaLayers` section. See Mask/Layer Functions on page 145.

Equivalent Layers

In the Virtuoso environment, the technology file continues to support layers which are considered equivalent in the `equivalentLayers()` construct in the `layerRules()` section of the technology file. This section is used to define different layer purpose pairs which represent the same type of material. However, this section can be used to define layers which connect by overlap instead of through a via. An example of this is local interconnect layers such as where `li` connects directly to `metal1` and `Vdd` is a layer which has the same mask number as `metal1`.

Translation of Technology File Stream Rules

Translation rules in your CDB technology file are written to a separate layer mapping file called *`libName`*`.layermap`, located in the same directory as the technology file, in the Virtuoso environment format shown above.

In CDB, stream translation rules can exist in both the technology file and in one or more separate layer mapping files (for different vendors or customers). CDB does not have a convention for naming layer mapping files.

When creating the stream layer mapping file from the technology file, check both layer mapping files and delete or rename them as required to ensure that you are using the correct stream layer rules in the Virtuoso Studio design environment version of the library.

Differences Between PIPO and XStream

For information about the differences in the functionality and use model of PIPO and XStream, see Migrating from PIPO to XStream on page 389.

Mask/Layer Functions

The `layerFunctions` class name has been changed to `functions`. It is recommended that all layers have a `maskNumber` defined. When not set, the `maskNumber` attribute of a layer defaults to `oacUnsetMaskNumber = 0xffffffffu`.

Layer mask numbers must be fully specified. Meaning a layer set in which part of the layers have mask number set and part of them do not will not be accepted. This would include layers that are derived from iccLayers rules or prRules.

Defining the layer mask numbers in the technology file `functions` section, gives explicit information about which layers are adjacent to each other. Layer function are used by the extractors of applications such as the XL layout editor to determine interconnecting layers. The *Create Wire* command also uses the `functions` list to determine routing layers and adjacent vias. Specify mask numbers for all inter-connect layers in the technology file.

```
functions(
 ;( layer      function   [maskNumber])
;( -----       ---------------------)
( diff          "ndiff    "1    )
( ndiff         "ndiff    "2    )
( pdiff         "pdiff    "3    )
( poly          "poly     "4    )
( cont          "cut      "5    )
( m1            "metal    "6    )
( via1          "cut      "7    )
( m2            "metal    "8    )
( via2          "cut      "9    )
( m3            "metal    "10   )
( via3          "cut      "11   )
( m4            "metal    "12   )
( via4          "cut      "13   )
( m5            "metal    "14   )
) ;functions
```

Valid Values: `cut`, `li`, `metal`, `ndiff`, `pdiff`, `nplus`, `pplus`, `nwell`, `pwell`, `poly`, `diff`, `recognition`, `other`, `unknown`

If the `layerFunctions` class is not defined in CDB, mask numbers and function of layers are mapped according to the `viaLayers` section.

If the material number is not defined, the following layer types are grouped according to their name in the following order:

1. well
2. implant
3. diffusion
4. poly

All layers that can not be classified are specified at the end of the list or without a number.

`prRules` or `iccRules` can reset the mask numbers of layers, however the order of layers will be kept when the viaLayers section is processed.

In past releases a user defined layer property such as `iccMaskNumber` could be used for user defined purposes. Mask numbers are now handled with a native OpenAccess construct. To avoid conflicting information, this type of property is no longer supported and should be removed from technology data.

Layer CDB Function to OpenAccess Material Mapping

OpenAccess enum name (#)	CDB enum name (#)	ASCII technology file name
oacMaterialOther (0)	techcUnknownLayerFunction (0)	“unknown”
oacMaterialNWell (1)	techcLayerNwell (1)	“nwell”
oacMaterialPWell (2)	techcLayerPwell (2)	“pwell”
oacMaterialPImplant (6)	techcLayerNplus (3)	“nplus” (“nimplant”)
oacMaterialPImplant (6)	techcLayerPplus (4)	“pplus” (“pimplant”)
oacMaterialNDiff (3)	techcLayerNdiff (5)	“ndiff”
oacMaterialPDiff (4)	techcLayerPdiff (6)	“pdiff”
oacMaterialPoly (7)	techcLayerPoly (7)	“poly”
oacMaterialContactlessMetal (10)	techcLayerLI (8)	“li”
oacMaterialMetal (9)	techcLayerMetal (9)	“metal”
oacMaterialCut (8)	techcLayerCut (10)	“cut”
oacDiffMaterial	techcLayerDiffusion (11)	"diff"
oacRecognitionMaterial	techcLayerRecognition (12)	"recognition"

Manufacturing Resolution (layer)

Virtuoso Studio design environment ASCII Technology File Definition:

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

DFII on CDB ASCII Technology File Definition:

```
layerRules(
   layerMfgResolutions(
   ( tx_layer g_value ) ... )
) ;layerRules
```

OpenAccess Constraint:

Class: `oaTech`
`::getDefaultManufacturingGrid()`
Class: `oaPhysicalLayer`
`::resetManufacturingGrid()`

Layer Routing Grids

Previously, routing grid information was specified as a layer attribute. In the Virtuoso Studio design environment, routing grid information is specified as a constraint. See Single Layer Routing Grids.

Preferred Routing Direction

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `routingDirections`

```
constraintGroups(
("foundry"
   routingDirections(
      (tx_layer      "horizontal")
      (tx_layer      "vertical")
      (tx_layer      "leftDiag")
      (tx_layer      "rightDiag")
      (tx_layer      "none")
   ) ;routingDirections
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

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

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacPreferredRoutingDirection`
Value Type: `oaIntValue` representing the four possible directions, vertical, horizontal, 45 degrees, and 135 degrees:
`oacPrefRoutingDirEnum`,
`oacNotApplicablePrefRoutingDir`,
`oacNonePrefRoutingDir`,
`oacHorzPrefRoutingDir`,
`oacVertPrefRoutingDir`,
`oacLeftDiagPrefRoutingDir`, and
`oacRightDiagPrefRoutingDir`.

Previously, the routing directions were specified as layer attributes. In the Virtuoso Studio design environment, routing direction information is specified as a constraint. Because routers can specify an override to the default preferred routing direction within a defined region, the routing direction information specified as a constraint meets this requirement of routers.

Current Density Rules

The current density rules in the `electricalRules` section of the CDB technology file are mapped to attributes of a layer in the Virtuoso environment.

Peak AC Current Density

Virtuoso Studio design environment ASCII Technology File Definition:

Layer rule name: `peakACCurrentDensity`

```
layerRules(
   currentDensity(
      ( peakACCurrentDensity tx_layer g_value )
      ...
      ) ;currentDensity
   currentDensityTables(
      ( peakACCurrentDensity tx_layer
      ( ("frequency" nil nil ["width"|"cutArea" nil nil] )
          [g_defaultValue ] )
      ( g_table )
      ...
       );peakACCurrentDensity
   ) ;currentDensityTables
) ;layerRules
```

DFII on CDB ASCII Technology File Definition:

Rule name: `ACCURRENTDENSITY PEAK` and `peakACCurrentDensity`

```
electricalRules(
   tableCharacterizationRules(
      ("ACCURRENTDENSITY PEAK" tx_layer
      ( "FREQUENCY"  (g_freqValue1 g_freqValue2...)   nil )
      "WIDTH" (g_widthValue1 g_widthValue2 )... nil
      ( g_table )
      )
   )
)

electricalRules(
   tableCharacterizationRules(
      ("peakACCurrentDensity" tx_layer
      ( "FREQUENCY"  (g_freqValue1 g_freqValue2...)   nil )
      "WIDTH" (g_widthValue1 g_widthValue2 )... nil
      ( g_table )
      )
   )
)
```

The LEF AC and DC current density in routing layers were mapped to the `tableCharacterizationRules` subsection in the `electricalRules` section of the technology file.

OpenAccess Constraint:

Class: `oaLayer`
Attribute: `PeakACCurrentDensity`
Value type: Table supports both `>=` and `>` look up.

Average AC Current Density

Virtuoso Studio design environment ASCII Technology File Definition:

Layer rule name: `avgACCurrentDensity`

```
layerRules(
   currentDensity(
      ( avgACCurrentDensity tx_layer g_value )
      ...
   ); currentDensity
   currentDensityTables(
      ( avgACCurrentDensity tx_layer
      ( ("frequency" nil nil ["width"|"cutArea" nil nil] )
          [g_defaultValue ] )
      ( g_table )
      ...
       ) ;avgACCurrentDensity
   ) ;currentDensityTables
) ;layerRules
```

DFII on CDB ASCII Technology File Definition:

Rule name: `ACCURRENTDENSITY AVERAGE` and `avgACCurrentDensity`

```
electricalRules(
   tableCharacterizationRules(
      ("ACCURRENTDENSITY AVERAGE" tx_layer
      ( "FREQUENCY"  (g_freqValue1 g_freqValue2...)   nil )
      "WIDTH" (g_widthValue1 g_widthValue2 )... nil
      ( g_table )
      )
   )
)
electricalRules(
   tableCharacterizationRules(
      ("avgACCurrentDensity" tx_layer
      ( "FREQUENCY"  (g_freqValue1 g_freqValue2...)   nil )
      "WIDTH" (g_widthValue1 g_widthValue2 )... nil
      ( g_table )
      )
   )
)
```

The LEF AC and DC current density in routing layers were mapped to the `tableCharacterizationRules` subsection in the `electricalRules` section of the technology file.

OpenAccess Constraint:

Class: `oaLayer`
Attribute: `AvgACCurrentDensity`
Value type: Table supports both >= and > look up.

RMS AC Current Density

Virtuoso Studio design environment ASCII Technology File Definition:

Layer rule name: `rmsACCurrentDensity`

```
layerRules(
   currentDensity(
      ( rmsACCurrentDensity tx_layer g_value )
      ...
   ); currentDensity
   currentDensityTables(
      ( rmsACCurrentDensity tx_layer
      ( ("frequency" nil nil ["width"|"cutArea" nil nil] )
          [g_defaultValue ] )
      ( g_table )
      ...
       ) :rmsACCurrentDensity
   ) ;currentDensityTables
) ;layerRules
```

DFII on CDB ASCII Technology File Definition:

Rule name: `ACCURRENTDENSITY RMS` and `rmsACCurrentDensity`

```
electricalRules(
   tableCharacterizationRules(
      ("ACCURRENTDENSITY RMS" tx_layer
      ( "FREQUENCY"  (g_freqValue1 g_freqValue2...)   nil )
      "WIDTH" (g_widthValue1 g_widthValue2 )... nil
      ( g_table )
      )
   )
)

electricalRules(
   tableCharacterizationRules(
      ("rmsACCurrentDensity" tx_layer
      ( "FREQUENCY"  (g_freqValue1 g_freqValue2...)   nil )
      "WIDTH" (g_widthValue1 g_widthValue2 )... nil
      ( g_table )
      )
   )
)
```

The LEF AC and DC current density in routing layers were mapped to the `tableCharacterizationRules` subsection in the `electricalRules` section of the technology file.

OpenAccess Constraint:

Class: `oaLayer`
Attribute: `RmsACCurrentDensity`
Value type: Table supports both >= and > look up.

Average DC Current Density

Virtuoso Studio design environment ASCII Technology File Definition:

Layer rule name: `avgDCCurrentDensity`

```
layerRules(
   currentDensity(
      ( avgDCCurrentDensity tx_layer g_value )
      ...
   ); currentDensity
   currentDensityTables(
      ( avgDCCurrentDensity tx_layer
      ( ("frequency" nil nil ["width"|"cutArea" nil nil] )
          [g_defaultValue ] )
      ( g_table )
      ...
       ) ;avgDCCurrentDensity
   ) ;currentDensityTables
) ;layerRules
```

DFII on CDB ASCII Technology File Definition:

Rule name: `DCCURRENTDENSITY AVERAGE` and `avgDCCurrentDensity`

```
electricalRules(
   tableCharacterizationRules(
      ("DCCURRENTDENSITY AVERAGE" tx_layer
      ( "FREQUENCY"  (g_freqValue1 g_freqValue2...)   nil )
      "WIDTH" (g_widthValue1 g_widthValue2 )... nil
      ( g_table )
      )
   )
)

electricalRules(
   tableCharacterizationRules(
      ("avgDCCurrentDensity" tx_layer
      ( "FREQUENCY"  (g_freqValue1 g_freqValue2...)   nil )
      "WIDTH" (g_widthValue1 g_widthValue2 )... nil
      ( g_table )
      )
   )
)
```

The LEF AC and DC current density in routing layers were mapped to the `tableCharacterizationRules` subsection in the `electricalRules` section of the technology file.

OpenAccess Constraint:

Class: `oaLayer`
Attribute: `AvgDCCurrentDensity`
Value type: Table supports both >= and > look up.

Cut Classes

For more information, see cutClasses in *Virtuoso Technology Data ASCII Files Reference*.

Virtuoso Studio design environment ASCII Technology File Definition:

Layer rule name: avgDCCurrentDensity

```
layerRules(
   cutClasses(
   [t_constraint_group_name]
   ; ( layer (className    ['numCuts numCuts]   ['minWidth] ['minLength]
   ; (width length) )... )
   ;( ------------------------------------------------------------------ )
     (lx_cutLayer (tx_name ['numCuts g_numCuts] ['minWidth] ['minLength]
      g_width | (g_width g_length))... )
   ) ;cutClasses
) ;layerRules
```

OpenAccess Constraint:

Constraint type: oaLayerConstraint (cut layers only)
Constraint name: cdcCutClass
Value type: oaDualIntValue: width, length, where length is optional
Parameters:

- oaStringValue:

  cdcClassNameConstraintParamType

- oaIntValue:

  oacNumCutsConstraintParamType

- oaBooleanValue:

  cdcViaWidthMinConstraintParamType: Default is false
  cdcViaLengthMinConstraintParamType: Default is false

LEF Syntax:

```
CUTCLASS className WIDTH viaWidth [LENGTH viaLength] [CUTS numCut];
```

Conversion of Technology File physicalRules Class

CDB	Virtuoso Studio design environment

physicalRules
  orderedSpacingRules
  spacingRules
  tableSpacingRules
  mfgGridResolution
  viaStackLimits
	constraintGroups
	  spacings
	  spacingTables
	  orderedSpacings
	  routingGrids
	  antennaModels
	  viaStackingLimits
	  interconnect(
	  validLayers
	  validVias

Changes:

Translation Processes on page 155

Mapping CDB Physical Technology File Rules to OpenAccess Constraints on page 156

Supported Constraints on page 160.

mfgGridResolution: See `controls` section, Manufacturing Grid on page 113

viaStackingLimits: Maximum Number of Via Stacking Constraint on page 299

New:

antennaModels: Antenna Models on page 265

validLayers: Valid Routing Layers Constraint on page 301

validVias: Valid Routing Vias Constraint on page 302

Translation Processes

The Virtuoso Studio design environment converts DFII on CDB technology file rules to constraints during any of the following processes:

- loading (compilation) of an ASCII technology file
- automatic updating performed by the software on already-loaded technology libraries to keep them up to date with the latest OpenAccess software

Mapping CDB Physical Technology File Rules to OpenAccess Constraints

Technology file constraints in the Virtuoso environment are designed to be interoperable with rule definitions on other applications on OpenAccess. In order to be interoperable, OpenAccess supports a set of well defined constraints and constraint definitions. For example, when correctly defined, the constraint `minArea` is a one layer constraint. If you attempt to defined `minArea` with two layers rather than one, the definition is inconsistent with the OpenAccess constraint definition.

There are several situations that can occur when mapping CDB technology file rules.

CDB Rules with Proper Syntax and Name

CDB Rules with Proper Syntax and Mapped Rule Name

CDB Rules with Proper Name and Incorrect Syntax

CDB Rules with a Unique Name

CDB Rules Using Layer Purpose Pairs

CDB Rules that are no Longer Supported

CDB Rules with Proper Syntax and Name

In this situation, the CDB rule name and definition comply with the OpenAccess definition of a constraint. These rules map correctly to OpenAccess constraint definitions and are interoperable with other applications on OpenAccess.

CDB Rules with Proper Syntax and Mapped Rule Name

In this situation, the CDB rule definition complies with the OpenAccess definition of a constraint and the rule name has been supported in past releases. For example, `minEnclosedArea` and `minHoleArea` were both used to specify the minimum area size limit for metal that enclosed an empty area. These rules names are mapped to the proper constraint name `minHoleArea` when compiling, or dumping and re-loading a technology file and are interoperable with other applications on OpenAccess. Names of rules that were previously used in the CDB technology data are listed in the mapping tables for each constraint.

If the CDB rule that is being mapped is an LPP rule, the name is mapped to the OpenAccess constraint name, but is only available for Virtuoso applications. See CDB Rules Using Layer Purpose Pairs.

CDB Rules with Proper Name and Incorrect Syntax

These are CDB rules that do not follow the OpenAccess constraint definition but have the same name as the OpenAccess constraint. These rules will not be mapped. For example, when correctly defined, the constraint `minArea` is a one layer constraint. If you attempt to defined `minArea` with two layers rather than one layer, the CDB rule definition is inconsistent with the OpenAccess constraint definition and the rule will fail to map or translate to the technology file.

In the Virtuoso Studio design environment, it is important in terms of sharing code across releases and databases that you define constraints/rules using the proper name, and definition.

CDB Rules with a Unique Name

These are CDB rules that have a unique name and can not be mapped to an OpenAccess constraint. These rules are stored as user defined constraints. These user defined constraints will only be accessible to Virtuoso applications and are not interoperable with other tools. User defined constraints are preserved in order to prevent Pcell code from breaking. These constraints are grouped in a separate section below the OpenAccess constraints. For example, the user defined spacing rule, `minMosWidth`, is defined as:

```
constraintGroups(
( "foundry"
;physical constraints
   spacings(
   ( minSpacing "METAL1" 0.2)
   ) ;spacings
   spacings(
   ( minMosWidth "METAL1" 0.3)
   ) ;spacings
) ;foundry
) ;constraintGroups
```

CDB Rules Using Layer Purpose Pairs

OpenAccess rules/constraints are only applicable to layers, not layer purpose pairs. Technology rules that use layer purpose pairs are stored as a private extension and are only available for Virtuoso applications.÷š

```
constraintGroups(
( "foundry"
;physical constraints
   spacings(
   ( minSpacing ("metal1" "drawing" ) 2.9 ) )
   ) ;spacings
   spacings(
```

In the example below, two CDB rules are mapped to the same constraint name. See CDB Rules with Proper Syntax and Mapped Rule Name. Since these specific rule names are mapped to the proper OpenAccess constraint name `minSameNetSpacing`, two constraints are created because they are LPP rules.

CDB rules:

```
constraintGroups(
( "foundry"
;physical constraints
   spacings(
   ( minStepEdgeLength ("metal1" "drawing" ) 2.9 ) )
   ( minEdgeLength ("metal1" "drawing" ) 1.9 ) )
   ) ;spacings
   spacings(
```

After translation to OpenAccess:

```
constraintGroups(
( "foundry"
;physical constraints
   spacings(
   ( minSameNetSpacing ("metal1" "drawing" ) 2.9 ) )
   ( minSameNetSpacing ("metal1" "drawing" ) 1.9 ) )
   ) ;spacings
   spacings(
```

A work-around is to remove duplicate rules such as these, is to dump the translated ASCII file and re-load the file. This will eliminate the duplicates. However, re-loading the technology file can cause the rules to be re-ordered.

After translation to OpenAccess:

```
constraintGroups(
( "foundry"
;physical constraints
   spacings(
   ( minSameNetSpacing ("metal1" "drawing" ) 2.9 ) )
   ( minSameNetSpacing ("metal1" "drawing" ) 1.9 ) )
   ) ;spacings
   spacings(
```

After dump and re-loading the technology file only one `minSameNetSpacing` remains.

```
constraintGroups(
( "foundry"
;physical constraints
   spacings(
   ( minSameNetSpacing ("metal1" "drawing" ) 1.9 ) )
   ) ;spacings
   spacings(
```

The CDB technology file is essentially defining the same rules with different values. In this situation, different rules could potentially be referenced in CDB and OpenAccess by applications such as DRD. Always check the technology file after translation for duplicate rules with the same value.

CDB Rules that are no Longer Supported

The only constraint names that can be mapped to an OpenAccess constraint are those listed in "Supported Constraints" on page 160. CDB rules that are no longer supported are treated as user defined rules and stored in Virtuoso private storage. See "User Defined Constraints" on page 100 for more information.

Note: `minButtedArea` and `minButtedClearance` are no longer supported and are stored as user defined constraints.

Supported Constraints

The mapping of CDB rules to the Virtuoso Studio design environment constraints can be found in the code *installDir*/dfII/group/techfile/src/gnTechRules/techRuleMap.cpp

Single Layer Spacing Constraints	
Gate Orientation	gateOrientation
Minimum Gate Extension Constraint	minGateExtension
Minimum Perimeter Constraint	minPerimeter
Minimum Area Constraint	minArea
Minimum Area Edge Length Constraint	minAreaEdgeLength
Minimum Size Constraint	minSize
Minimum Enclosed Area (Hole) Constraint	minHoleArea
Minimum Enclosed Area (Hole) Width Constraint	minHoleWidth
Minimum Cut Class Spacing	minCutClassSpacing
Minimum Cut Class Clearance	minCutClassClearance
Minimum Density Constraint	minDensity
Maximum Density Constraint	maxDensity
Minimum Step Edge Length Constraint	minStepEdgeLength
Minimum Diagonal Edge Length Constraint	minDiagonalEdgeLength
Manhattan Corner Enclosure or Spacing Constraint	minSpacing
Minimum Spacing Constraint	minSpacing
Minimum Center Spacing Constraint	minCenterToCenterSpacing
Minimum Diagonal Spacing Constraint	minDiagonalSpacing
Minimum Diagonal Width Constraint	minDiagonalWidth
Minimum Different Potential Spacing Constraint	minDiffNetSpacing
Minimum Space between Fill Pattern and Real Design Object Constraint	minFillToShapeSpacing
Minimum Enclosed Area (Hole) Width Constraint	minHoleWidth

Minimum Space between Fill Pattern and Real Design Object Constraint	`minFillToShapeSpacing`
Minimum Proximity (Influence) Spacing Constraint	`minInfluenceSpacing`
Minimum Proximity (Influence) Spacing Constraint for Protruding Wires or Stubs	`minStubInfluenceSpacing`
Minimum Spacing Constraint (sameNet)	`minSameNetSpacing`
Minimum Adjacent Via Spacing Constraint	`viaSpacing`
Minimum Via Spacing Constraint	`minViaSpacing`
Merge Allowed Spacing Constraint	`maxFilling`
Minimum Width Constraint	`minWidth`
Minimum Protrusion Width Constraint	`protrusionWidth`
Maximum Width Constraint	`maxWidth`
Minimum Length Constraint	`minLength`
Maximum Length Constraint	`maxLength`
Minimum Number of Cuts Constraint	`minNumCut`
Minimum Number of Cuts on Protrusion Constraint	`minProtrusionNumCut`
Allowed Shape Angles Constraint	`allowedShapeAngles`
Diagonal Shapes Allowed Constraint	`diagonalShapesAllowed`
Maximum Tap Spacing Constraint	`maxTapSpacing`
Minimum Notch Spacing	`minNotchSpacing`
Minimum End of Notch Spacing	`minEndOfNotchSpacing`
Minimum Wire Extension Constraint	`minWireExtension`
Minimum Boundary Extension Constraint	`minPRBoundaryExtension`
Keep prBoundary Shared Edges Constraint	`keepPRBoundarySharedEdges`
Minimum Boundary Interior Halo Constraint	`minPRBoundaryInteriorHalo`
Maximum Diagonal Edge Length Constraint	`maxDiagonalEdgeLength`
Redundant Via Setback Constraint	`redundantViaSetback`
Maximum Routing Distance Constraint	`maximumRoutingDistance`
Minimum Parallel Via Spacing Constraint	`minParallelViaSpacing`

Minimum Parallel Within Via Spacing Constraint	`minParallelWithinViaSpacing`
Minimum Parallel Span Spacing Constraint	`minParallelSpanSpacing`
	`minSameMetalSharedEdgeViaSpacing`
Minimum End of Line Spacing Constraint	`minEndOfLineSpacing`
Minimum End of Line Perpendicular Spacing Constraint	`minEndOfLinePerpSpacing`
Minimum Large Via Array Spacing Constraint	`minLargeViaArraySpacing`
Minimum Large Via Array Cut Spacing Constraint	`minLargeViaArrayCutSpacing`
Maximum Number of Edges with Minimum Edge Length Constraint	`maxNumMinEdges`
Minimum Distance Between Adjacent Sets of Edges with Minimum Edge Length Constraint	`minEdgeAdjacentDistance`
Minimum Length of Sets of Edges Adjacent to a Short Edge Constraint	`minEdgeAdjacentLength`
Minimum Rectangle Area Constraint	`minRectArea`
Minimum Corner to Corner Distance Constraint	`minCornerToCornerDistance`
Allowed Spacing Range Constraint	`allowedSpacingRanges`
Taper Halo Constraint	`taperHalo`
Corner Constraints	
Minimum Outside Corner Edge Length Constraint	`minOutsideCornerEdgeLength`
Minimum Inside Corner Edge Length Constraint	`minInsideCornerEdgeLength`
Minimum Inside Corner Overlap Constraint	`minInsideCornerOverlap`
Minimum Inside Corner Extension Constraint	`minInsideCornerExtension`
Single Layer Routing Grids	
Horizontal Routing Grid Pitch Constraint	`horizontalPitch`
Horizontal Routing Grid Offset Constraint	`horizontalOffset`
Vertical Routing Grid Pitch Constraint	`verticalPitch`
Vertical Routing Grid Offset Constraint	`verticalOffset`
Left Diagonal Routing Grid Pitch Constraint	`leftDiagPitch`

Left Diagonal Routing Grid Offset Constraint	`leftDiagOffset`
Right Diagonal Routing Grid Pitch Constraint	`rightDiagPitch`
Right Diagonal Routing Grid Offset Constraint	`rightDiagOffset`
Default Horizontal Routing Grid Pitch Constraint	`horizontalPitch`
Default Horizontal Routing Grid Offset Constraint	`horizontalOffset`
Default Vertical Routing Grid Pitch Constraint	`verticalPitch`
Default Vertical Routing Grid Offset Constraint	`verticalOffset`
Default Left Diagonal Routing Grid Pitch Constraint	`leftDiagPitch`
Default Left Diagonal Routing Grid Offset Constraint	`leftDiagOffset`
Default Right Diagonal Routing Grid Pitch Constraint	`rightDiagPitch`
Default Right Diagonal Routing Grid Offset Constraint	`rightDiagOffset`
Placement Grid Constraints	
Horizontal Placement Grid Pitch Constraint	`horizontalPitch`
Horizontal Placement Grid Offset Constraint	`horizontalOffset`
Vertical Placement Grid Pitch Constraint	`verticalPitch`
Vertical Placement Grid Offset Constraint	`verticalOffset`
Antenna Models	
Antenna Oxide1 Model Constraint	`antennaModels "default"`
Antenna Oxide2 Model Constraint	`antennaModels "second"`
Antenna Oxide3 Model Constraint	`antennaModels "third"`
Antenna Oxide4 Model Constraint	`antennaModels "forth"`
Two Layer Constraints	
Minimum Clearance Constraint	`minSpacing`
Maximum Spacing Constraint	`maxSpacing`
Minimum Clearance Constraint (sameNet)	`minSameNetSpacing`
Minimum Extension Constraint	`minExtension`

Minimum Opposite Extension Constraint	`minOppExtension`
Minimum Extension Edge Constraint	`minExtensionEdge`
Minimum Orthogonal Via Spacing Constraint	`minOrthogonalViaSpacing`
Coincident Extension Allowed Constraint	`'coincidentAllowed`
Minimum Overlap Constraint	`minOverlap`
Minimum Parallel Via Clearance Constraint	`minParallelViaSpacing`
Via Stacking Rule Constraint	`stackable`
Minimum Enclosure Constraint	`minEnclosure`
Maximum Enclosure Constraint	`maxEnclosure`
Keep Shared Edges Constraint	`keepSharedEdges`
Allowed Clearance Range Constraint	`allowedSpacingRanges`
Minimum Via Clearance Constraint	`minViaSpacing`
Three Layers Spacing Constraints	
Minimum Enclosure in Direction of Touching Layer Constraint	`minTouchingDirEnclosure`
Minimum Spacing in Direction of Touching Layer Constraint	`minTouchingDirSpacing`
Minimum Spacing Over Layer Constraint	`minSpacingOver`
Dummy Poly Extension Constraint	`dummyExtension`
Constraints Associated with the Technology Database	
Maximum Number of Via Stacking Constraint	`viaStackingLimits`
Valid Routing Layers Constraint	`validLayers`
Valid Routing Vias Constraint	`validVias`
Global Distance Measure	`distanceMeasure`

Mapping Table Conventions

The following table describes the naming conventions used in mapping the CDB rules to the Virtuoso Studio design environment technology file constraints.

Virtuoso Studio design environment ASCII Technology File Definition:	Listed is the Virtuoso Studio design environment constraint name, and the ASCII technology file definition. The OpenAccess database constraint can be accessed (using the Virtuoso environment constraint name) with C or SKILL APIs (CDBA).
DFII on CDB ASCII Technology File Definition:	Listed is the CDB rule name(s), and the ASCII technology file definition. In some case, more than one rule name is mapped to a Virtuoso environment constraint name. The CDB technology file rule can be accessed using C or SKILL APIs (CDBA).
OpenAccess Constraint:	The OpenAccess 2.2 database constraint representation. The OpenAccess database constraint can be accessed by name using C++.
LEF Syntax:	The LEF syntax that is mapped to the Virtuoso Studio design environment technology file constraint and constraint group.

Single Layer Spacing Constraints

The following describes the mapping of the single layer rules section of the DFII on CDB technology file to the Virtuoso Studio design environment technology file.

Gate Orientation

All gates below a certain width must be created in the same orientation, horizontal or vertical. This constraint applies to a derived layer that specifies the gate. If no width is specified, the constraint applies to gates of any width.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `gateOrientation`

```
constraintGroups(
("foundry"
   spacings(
      ( gateOrientation tx_derivedLayer 'width g_width "horizontal" )
      ( gateOrientation tx_derivedLayer 'width g_width "vertical" )
      ( gateOrientation tx_derivedLayer 'width g_width "any" )
   );spacings
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacGateOrientation`
Value type: `oaIntValue - oaGateOrientationType`, `oacVerticalGateOrientation`, `oacHorizontalGateOrientation`, `oacAnyGateOrientation`

Minimum Gate Extension Constraint

This constraint specifies the poly gate extension over diffusion when an L-shaped oxide is within a given distance. For more information, see `minGateExtension` in *Virtuoso Technology Data ASCII Files Reference*.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minGateExtension`

```
constraintGroups(
("foundry"
   orderedSpacings(
      ( minGateExtension tx_layer1 tx_layer2 'distance g_distance
       'length g_length g_extension )
   );orderedSpacings
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Constraint type: `oaLayerPairConstraint` (non-symmetric)
Constraint name: `cdcMinGateExtension`
Value type: `oaIntValue`: minimum extension
Parameters: `oaIntValue`:
`oacDistanceConstraintParamType`
`cdcMinJogLengthConstraintParamType`

LEF Syntax:

NA

Minimum Perimeter Constraint

This constraint specifies the perimeter of a shape. For more information, see `minPerimeter` in *Virtuoso Technology Data ASCII Files Reference*.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minPerimeter`

```
constraintGroups(
("foundry"
   spacings(
      ( minPerimeter tx_layer g_perimeter )
   ) ;spacings
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Constraint Type: `oaLayerConstraint`
Constraint Name: `cdcMinPerimeter`
Value Type: `oaIntValue`: minimum perimeter
Parameters: None

LEF Syntax:

NA

Minimum Area Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minArea`

```
constraintGroups(
("foundry"
   spacings(
      ("minArea" tx_layer g_area)
      ...
   );spacings
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `minArea`

```
physicalRules(
   spacingRules(
      ("minArea" tx_layer g_area)
   ) ;spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinArea`
Value type: `oaIntValue`

Minimum Area Edge Length Constraint

This constraint specifies the perimeter of a shape. For more information, see `minAreaEdgeLength` in *Virtuoso Technology Data ASCII Files Reference*.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minAreaEdgeLength`

```
constraintGroups(
("foundry"
   spacings(
     (minAreaEdgeLength tx_layer
       ['exceptEdgeLength g_length]
       ['exceptMinSize (g_width g_length)]]
      g_area)
   );spacings
);foundry
);constraintGroups
```

OpenAccess Constraint:

Constraint type: `oaLayerConstraint`
Constraint name: `cdcMinAreaEdgeLength`
Value: `oaIntValue`: minimum area
Parameters:

- `oaIntValue`:

  `cdcExceptEdgeLengthConstraintParamType`: minimum edge length
  `oacWidthConstraintParamType`

- `oaDualIntValue`:

  `cdcExceptMinSizeConstraintParamType`

LEF Syntax:

```
AREA minArea [
  [EXCEPTEDGELENGTH edgeLength]
  [EXCEPTMINSIZE minWidth minLength];
```

Minimum Size Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minSize`

```
constraintGroups(
("foundry"
   spacings(
      ("minSize" tx_layer (g_width g_length) [ ( g_width g_length ) ...] )
      ...
   );spacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `minSize`

```
physicalRules(
   spacingRules(
      ("minSize" tx_layer (g_width g_length) )
   );spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinSize`
Value type: `oaBoxArrayValue`

Minimum Enclosed Area (Hole) Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minHoleArea`

```
constraintGroups(
("foundry"
   spacings(
      "minHoleArea" tx_layer g_area)
      ...
   );spacings
);foundry
);constraintGroups

constraintGroups(
("foundry"
   spacingTables(
      ("minHoleArea" tx_layer
         ( ("edge2edgeWidth" nil nil) [ g_defaultValue ])
         (l_table)
      )
   );spacingTables
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule names: `minHoleArea` and `minEnclosedArea`

```
physicalRules(
   spacingRules(
      ("minHoleArea" tx_layer g_area)
   );spacingRules
);physicalRules
```

The LEF statement, `MINENCLOSEDAREA` previously mapped to the `minEnclosedArea` rule in the `spacingRules` section. When the "`area`" construct was used, this specified the minimum area size limit for metal that enclosed an empty area. When `"width"` was specified, the rule mapped to the `tableSpacingRules` section.

```
physicalRules(
   spacingRules(
      ("minEnclosedArea" tx_layer g_area)
   );spacingRules
);physicalRules

physicalRules(
   tableSpacingRules(
      ("minEnclosedArea" tx_layer)
      ("width" nil nil)
      (l_table)
   );tableSpacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinEnclosedArea` with 1 parameter; `width`
Value type: `oaIntValue`, `oaInt1DTblValue` (`width`)

Minimum Enclosed Area (Hole) Width Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minHoleWidth`

```
constraintGroups(
("foundry"
   spacings(
      ("minHoleWidth" tx_layer g_width)
      ...
   );spacings
);foundry
);constraintGroups

constraintGroups(
("foundry"
   spacingTables(
      ("minHoleWidth" tx_layer
      ( ("edge2edgeWidth" nil nil) [ g_defaultValue ] )
      (l_table)
      )
   );spacingTables
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `minHoleWidth`

```
physicalRules(
   spacingRules(
      ("minHoleWidth" tx_layer g_width)
   );spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinEnclosedSpacing`
Value type: `oaIntValue`, `oaInt1DTblValue(width)`

Minimum Cut Class Spacing

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

For the same cut layer, there can be up to two `minCutClassSpacing` constraints defined, one with either `'sameNet` or `'sameMetal` and the other constraint without the `'sameNet` or `'sameMetal`. Only `viaSpacing`, `minParallelWithinViaSpacing` and `minSameMetalSharedEdgeViaSpacing` cut spacing constraints should be used in conjunction with the `minCutClassSpacing` constraint.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minCutClassSpacing`

```
constraintGroups(
("foundry"
   spacingTables(
      (minCutClassSpacing tx_cutLayer
      ( ( "cutClass" (g_index...) nil "cutClass" nil|(g_index...) nil)
      ['sameNet|'sameMetal] ['paraOverlap [g_overlap]] [g_default])
      (g_table)
      )
   );spacingTables
);foundry
);constraintGroups
```

where an ordered list of row indexes must be added for the row, and must be present in the table.

OpenAccess Constraint:

Constraint type: `oaLayerConstraint`
Constraint name: `cdcMinCutClassSpacing`
Value: `oaInt2DTblValue`
Parameters:

- `oaIntValue enum`:

  `oacConnectivitiyTypeConstraintParamType`: same net, same metal only
  `cdcAnyConnectvityType` (default)
  `cdcSameNetConnectivityType`
  `oacContiguousShapesConnectivityType`
  `cdcParallelRunLengthConstraintParamType`: Default value is 0. The possible values are:
  -1: The constraint applies only when there is no parallel run length
  0: The constraint applies when there is a >= 0 parallel run length
  1: The constraint applies when there is a > 0 parallel run length
  +integer: The constraint applies when there is a >= +integer parallel run length

  `oaArrayValueArray` of `oaDualIntValue`: `cdDualIntValueArray`
  `cdcCutClassListConstraintParamType`: cut class pairs

- `oaInt2DTblValue`:

  `cdcCutClassCenterToCenterConstraintParamType`: index cut class (width, length) value `cdSpacingMeasureTypeEnum` of values `cdcEdgeToEdgeSpacingMeasureType` (the default) and `cdcCenterToCenterSpacingMeasureType`

LEF Syntax:

```
SPACINGTABLE [DEFAULT defaultCutSpacing]
  [SAMENET|SAMEMETAL] [CENTERTOCENTER {{className1|ALL} TO {className2|ALL}}...]
  CUTCLASS
    {{className1|ALL} [SIDE|END]}...
    {{className2|ALL} [SIDE|END] {-|cutSpacing}{-|cutSpacing}...}...;
```

Minimum Cut Class Clearance

For more information, see minCutClassSpacing in *Virtuoso Technology Data ASCII Files Reference*.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minCutClassSpacing`

```
constraintGroups(
("foundry"
   spacingTables(
      (minCutClassSpacing tx_cutLayer1 tx_cutLayer2
      ( ( "cutClass" (g_index...) nil "cutClass" (g_index...) nil)
      ['sameNet|'sameMetal] ['paraOverlap [g_overlap]] [g_default])
      (g_table)
      )
   );spacingTables
);foundry
);constraintGroups
```

where an ordered list of row indexes must be added for the row, and must be present in the table.

OpenAccess Constraint:

Constraint type: `oaLayerPairConstraint` (non-symmetric)
Constraint name: `cdcMinCutClassClearance`
Value: `oaInt2DTblValue`
Parameters:

- `oaIntValue enum`:

  `oacConnectivitiyTypeConstraintParamType`: same net, same metal only
  `cdcAnyConnectvityType` (default)
  `cdcSameNetConnectivityType`
  `oacContiguousShapesConnectivityType`
  `cdcParallelRunLengthConstraintParamType`: Default value is 0. The possible values are:
  -1: The constraint applies only when there is no parallel run length
  0: The constraint applies when there is a >= 0 parallel run length
  1: The constraint applies when there is a > 0 parallel run length
  +integer: The constraint applies when there is a >= +integer parallel run length

  `oaArrayValueArray` of `oaDualIntValue`: `cdDualIntValueArray`
  `cdcCutClassListConstraintParamType`: cut class pairs

- `oaInt2DTblValue`:

  `cdcCutClassCenterToCenterConstraintParamType`: index cut class (width, length) value `cdSpacingMeasureTypeEnum` of values `cdcEdgeToEdgeSpacingMeasureType` (the default) and `cdcCenterToCenterSpacingMeasureType`

LEF Syntax:

```
SPACINGTABLE [DEFAULT defaultCutSpacing]
  [SAMENET|SAMEMETAL]
  LAYER secondLayerName
  [CENTERTOCENTER {{className1|ALL} TO {className2|ALL}}...]
  CUTCLASS
    {{className1|ALL} [SIDE|END]}...
    {{className2|ALL} [SIDE|END] {-|cutSpacing}{-|cutSpacing}...}...;
```

Minimum Density Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minDensity`

```
constraintGroups(
("foundry"
   spacings(
      ("minDensity" tx_layer g_density)
      ...
   );spacings
);foundry
);constraintGroups

constraintGroups(
("foundry"
   spacingTables(
      ("minDensity" tx_layer
      ( ("step" nil nil) [ g_defaultValue ] )
      (g_table)
      )
   );spacingTables
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule names: `minDensity` and `minimumDensity`

```
physicalRules(
   spacingRules(
      ("minDensity" tx_layer g_density)
   );spacingRules
);physicalRules


physicalRules(
   spacingRules(
      ("minimumDensity" tx_layer g_density)
   );spacingRules
);physicalRules
```

The LEF metal filling value, `MINIMUMDENSITY previously` mapped to the `minimumDensity` rule in the `spacingRules` subsection.

Density is intended to represent the minimum percent density for a layer.

The step size is used to define the area to be checked. The area is set to be (2 * step) ^2. The entire chip is checked by examining this window set at step sized intervals in the x and y directions. The step size is almost half the foundry density tolerance.

The CDB rules `densityCheckStep` and `densityCheckWindow` both map to the step value in the `minDensity` constraint, with the `densityCheckWindow` taking precedence over the `densityCheckStep`.

If `densityCheckStep` or `densityCheckWindow` exist, `minDensity` is converted to a `spacingTables` constraint. Otherwise, the `minDensity` rule is `spacings` constraint.

Note: Scalar spacing rule `minDensity` and `minimumDensity`, when combined with `densityCheckStep` and/ or `densityCheckWindow` rules, are converted to a `spacingTables` constraint. Once it is converted, it is no longer appropriate to query the constraint using the scalar rule API `techGetSpacingRule`. Use the table API, `techGetSpacingRuleTable`.

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinDensity`
Value type: `oaFltValue`, `oaFltIntTblValue`(step)

Maximum Density Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `maxDensity`

```
constraintGroups(
("foundry"
   spacings(
      ("maxDensity" tx_layer g_density )
       ...
   );spacings
);foundry
);constraintGroups

constraintGroups(
("foundry"
   spacingTables(
      ("maxDensity" tx_layer
         (
         ( ("step" nil nil) [ g_defaultValue ] )
         (l_table)
         )
      )
   ) ;spacingTables
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule names: `maxDensity` and `maximumDensity`

```
physicalRules(
   spacingRules(
      (maxDensity tx_layer g_density)
   );spacingRules
);physicalRules

physicalRules(
   spacingRules(
      (maximumDensity tx_layer g_density)
   );spacingRules
);physicalRules
```

The LEF metal filling value, `MAXIMUMDENSITY` mapped to the `maximumDensity` rule in the `spacingRules` subsection.

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMaxDensity`
Value type: `oaFltValue`, `oaFltIntTblValue`(step)

Minimum Step Edge Length Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minStepEdgeLength`

```
constraintGroups(
("foundry"
   spacings(
      ("minStepEdgeLength" tx_layer [g_lengthSum] g_length) )
      ...
   );spacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

In OpenAccess 2.2, three minimum edge length constraints have been added to represent different types of edges: `minOutsideCornerEdgeLength`, `minInsideCornerEdgeLength`, and `minStepEdgeLength`.

Rule names: `minStep` and `minEdgeLength`.

```
physicalRules(
   spacingRules(
      ("minEdgeLength" tx_layer [g_lengthSum] g_length) )
   );spacingRules
);physicalRules
```

In LEF 5.5, the statement `MINSTEP` specified the minimum step size (or shortest edge length). In this rule, distance was a float in units of `um` and it specifies the minimum step size.

```
physicalRules(
   spacingRules(
      ("minStep" tx_layer g_length)
   );spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinEdgeLength`
Value type: `oaIntValue` with optional `oaConstraintParamArray` to store `oacLengthSumConstraintParamType`

Minimum Diagonal Edge Length Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minDiagonalEdgeLength`

```
constraintGroups(
("foundry"
   spacings(
      ("minDiagonalEdgeLength" tx_layer g_length)
      ...
   );spacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `minDiagonalEdgeLength`

```
physicalRules(
   spacingRules(
      ("minDiagonalEdgeLength" tx_layer g_length)
   );spacingRules
);physicalRules
```

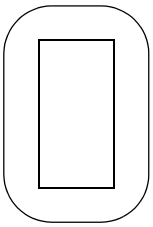
OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinDiagonalEdgeLength`
Value type: `oaIntValue` with optional `oaConstraintParamArray` to store `oacLengthSumConstraintParamType`

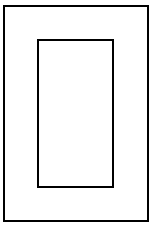
Manhattan Corner Enclosure or Spacing Constraint

Spacing constraints define a minimum spacing at any angle around a particular shape. This means that the spacing halo round a shape is an arc at the corner as defined in figure A. Constraints have been enhanced to support a larger spacing at the corners and define the spacing as the manhattan distance from the corner as defined in figure B.

A. Euclidean spacing halo



B. Manhattan spacing halo



Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minSpacing` Value: `manhattan`

```
constraintGroups(
("foundry"
   spacings(
      ( minSpacing tx_layer g_spacing ['manhattan] )
   ) ;spacings
   orderedSpacings(
      ( minEnclosure tx_layer1 tx_layer2 g_enclosure ['manhattan] )
      ( minSpacing tx_layer1 tx_layer2 g_spacing ['manhattan] )
   ) ;orderedSpacings
   spacingTables(
       ("minSpacing" tx_layer1 [tx_layer2]
      (("width" nil nil ["length" nil nil] ))
       ( g_table )
      ['manhattan] )
   ) ;spacingTables
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Parameter: `oaConstraintParam`: `oaEuclidean`
Value type: `oaBoolean`. If true, distance is measured in Euclidean directions. If false the direction is measured in Manhattan directions. Default is true.
Supported constraints: `oacMinSpacing`, `oacMinClearance`, `oacMinExtension`, `oacMinTouchingDirectionExtension`, and `oacMinTouchingDirectionClearance`

Minimum Spacing Constraint

This constraint specifies the minimum spacing between any two adjacent geometries on a specified layer. The spacing can be measured either in manhattan or euclidean metric (see Manhattan Corner Enclosure or Spacing Constraint). The value of the constraint can be based either on the width of the wider of the two shapes, on the width of the wider of the two shapes plus their parallel run length, on the width of both the shapes, or on the width of both the shapes plus their parallel run length. An optional parameter determines the spacing direction, horizontal or vertical, in which the constraint applies. Another optional parameter determines whether the constraint applies to power and ground nets.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ( minSpacing tx_layer ['horizontal | 'vertical | 'any]
        ['sameNet ['PGNet] | 'sameMetal] g_space
         ['manhattan] )
      ...
   ) ;spacings
) ;foundry
)
constraintGroups(
("foundry"
   spacingTables(
      ( minSpacing tx_layer
         ( ("width" nil nil ["length"|"width" nil nil] ) ['horizontal |
           'vertical | 'any] ['sameNet ['PGNet] | 'sameMetal]
            [g_defaultValue] )
            (g_table)
            ['soft | 'hard] ['coincidentAllowed] ['manhattan] ['ref t_ref]
            ['description t_description]
      )
      ( minSpacing tx_layer
         ( ("twoWidths" nil nil "length" nil nil) ['horizontal |
           'vertical | 'any] ['sameNet ['PGNet] | 'sameMetal]
            [g_defaultValue] )
            (g_table) ['manhattan]
      )
      )
   ) ;spacingTables
) ;foundry
) ;constraintGroups
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinSpacing`
Value type: `oaIntValue, oaInt1DTblValue(`*`width`*`), oaInt2DTblValue(`*`width`*`, `*`length`*`)`
Parameter: `oacWidthLengthTableTypeConstraintParamType`, `oaIntValue`, `oaWidthLengthTableType value oacTwoWidthParallelRunLengthTableType`

Spacing table supports both >= and > semantic look up.

Minimum Center Spacing Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minCenterToCenterSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ("minCenterToCenterSpacing" tx_layer g_value)
      ...
   );spacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `minCenterToCenterSpacing`

```
physicalRules(
   spacingRules(
      ("minCenterToCenterSpacing" tx_layer g_value)
   );spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinCenterToCenterSpacing`
Value type: `oaIntValue`

Minimum Diagonal Spacing Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minDiagonalSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ("minDiagonalSpacing" tx_layer g_value)
      ...
   );spacings
);foundry
);constraintGroups

constraintGroups(
("foundry"
   spacingTables(
      ("minDiagonalSpacing" tx_layer
         (
         ( ("width" nil nil ["length" nil nil] ) [ g_defaultValue ])
         (l_table)
         )
      )
   );spacingTables
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `minDiagonalSpacing`

```
physicalRules(
   spacingRules(
      ("minDiagonalSpacing" tx_layer g_value)
   );spacingRules
);physicalRules
```

OpenAccess Constraint: object

Class: `oaLayerConstraint`
Constraint type: `oacMinDiagonalSpacing`
Value type: `oaIntValue`, `oaInt1DTblValue(width)`, `oaInt2DTblValue(width, length)`

Minimum Diagonal Width Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minDiagonalWidth`

```
constraintGroups(
("foundry"
   spacings(
      ("minDiagonalWidth" tx_layer g_width)
      ...
   );spacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `minDiagonalWidth`

```
physicalRules(
   spacingRules(
      ("minDiagonalWidth" tx_layer g_wdith)
   );spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinDiagonalWidth`
Value type: `oaIntValue`,

Minimum Different Potential Spacing Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minDiffNetSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ("minDiffNetSpacing" tx_layer g_value)
      ...
   );spacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `minDiffNetSpacing`

```
physicalRules(
   spacingRules(
      ("minDiffNetSpacing" tx_layer g_value)
   );spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinDiffPotentialSpacing`
Value type: `oaIntValue`,

Minimum Space between Fill Pattern and Real Design Object Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minFillToShapeSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ("minFillToShapeSpacing" tx_layer g_value)
      ...
   );spacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `minFillToShapeSpacing` and `fillActiveSpacing`

```
physicalRules(
   spacingRules(
      ("minFillToShapeSpacing" tx_layer g_value)
   );spacingRules
```

The LEF metal filling value, `FILLACTIVESPACING` mapped to the `fillActiveSpacing` rule in the `spacingRules` subsection.

```
physicalRules(
   spacingRules(
      ("fillActiveSpacing" tx_layer g_value)
   );spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinFillPatternSpacing`
Value type: `oaIntValue`

Minimum Proximity (Influence) Spacing Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minInfluenceSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ("minInfluenceSpacing" tx_layer g_value)
      ...
   );spacings
);foundry
);constraintGroups


constraintGroups(
("foundry"
   spacingTables(
      ("minInfluenceSpacing" tx_layer
         (
         ("width" nil nil ["distance" nil nil] ) [ g_defaultValue ] )
         (l_table)
         )
      )
   )
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `minInfluenceSpacing`

```
physicalRules(
   tableSpacingRules(
      ("minInfluenceSpacing" tx_layer )
      ("width" nil nil ["distance" nil nil])
      (l_table)
   );tableSpacingRules
);physicalRules
```

The LEF statement `SPACINGTABLE PARALLELRUNLENGTH` previously mapped to `minSpacing` in the `tableSpacingRules` subsection.

The LEF statement `SPACINGTABLE INFLUENCE` previously mapped to "`minSpacingInfluence`" under the `tableSpacingRules` subsection.

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinProximitySpacing` with 2 parameters, `width` and `distance`.
Value type: `oaIntValue, oaInt1DTblValue(width), oaInt2DTblValue(width, distance)`

Spacing table supports both >= and > look up.

Minimum Proximity (Influence) Spacing Constraint for Protruding Wires or Stubs

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minStubInfluenceSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ("minStubInfluenceSpacing" tx_layer g_spacing
      ...
   );spacings
);foundry
);constraintGroups


constraintGroups(
("foundry"
   spacingTables(
      ("minStubInfluenceSpacing" tx_layer
         (
         ( ("width" nil nil ["distance" nil nil] ) )
         (g_table)
         )
      )
   ) ; spacingTables
);foundry
);constraintGroups
```

LEF spacing semantic has “>=”. The spacing table influence is not mapped to this constraint. The influence halo of the larger shape is given by the `minSpacing` constraint.

DFII on CDB ASCII Technology File Definition:

Rule name: `minStubInfluenceSpacing`

```
physicalRules(
   spacingRules(
      ("minStubInfluenceSpacing" tx_layer g_value)
   );spacingRules
);physicalRules


physicalRules(
   spacingRules(
      ( minSpacingRange   tx_layer ("minSpacingValue RANGE minWidth maxWidth
INFLUENCE infvalue")
   ) ;spacingRules
) ;physicalRules
```

`minSpacingRange` with the `INFLUENCE` keyword maps to a 2D table `minStubInfluenceSpacing` constraint.

"`RANGE stubMinWidth stubMaxWidth`" are ignored with warning message.

`minWidth` is used as rows and *`influenceValue`* is used as columns. The values are *`minSpacingValues`*

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinProtrudedProximitySpacing`
Value type: `oaIntValue, oaInt1DTblValue(`*`width`*`), oaInt2DTblValue(width, distance)`

Spacing table supports both >= and > look up.

Minimum Spacing Constraint (sameNet)

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minSameNetSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ("minSameNetSpacing" tx_layer g_value)
      ...
   spacingTables(
      ( "minSameNetSpacing" tx_layer
      (( "width" nil nil ))
      ( g_table )
      )
   ) ;spacingTables
   );spacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule names: `minSameNetSpacing`, `minNotch` and `sameNet`

```
physicalRules(
   spacingRules(
      ("minSameNetSpacing" tx_layer g_value)
   )
)


physicalRules(
   spacingRules(
      ("minNotch" tx_layer g_value)
   );spacingRules
);physicalRules
```

The LEF `SPACING` statement previously mapped same net spacing rules and stacked vias. Without the `STACK` statement, same net spacing and was mapped to the `sameNet` rule in the `spacingRules` subsection.

```
physicalRules(
   spacingRules(
      ("sameNet" tx_layer g_value)
   );spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinSameNetSpacing`
Value type: `oaIntValue`

Minimum Adjacent Via Spacing Constraint

This constraint specifies the required minimum distance between adjacent via cuts and depends on the number of adjacent cut shapes within the specified distance. A via cut is adjacent if it is within the specified distance from another cut in any direction including a 45-degree angle. Optional parameters can be used to determine the following:

- Whether the cut spacing is to be measured edge-to-edge or center-to-center
- Whether the constraint applies to shapes having any connectivity, to shapes having only same-net connectivity, or only to cut shapes on the same metal
- Whether the constraint applies to vias on power and ground nets

The constraint can optionally apply only if the cut shapes are either on the same net or are overlapped by the same metal shape or are overlapped by the same metal shape above and below which cover the overlap area between the cuts. The number of neighboring cuts can also be specified.

This constraint supports 32 nm design rules. The cutClass constraint must be defined before this constraint. For more information, see `viaSpacing` in *Virtuoso Technology Data ASCII Files Reference*.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `viaSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ( viaSpacing tx_cutLayer ['centerToCenter]
      ['sameNet|'sameMetal|'sameMetalOverlap|'sameVia]
      ['exceptSamePGNet]
      ['cutClass g_width | (g_width g_length) | t_name]
      ['exactAligned g_numAlignedCuts ['exactSpacing]]
      'numCuts g_numCuts 'distance g_distance g_spacing )
   ) ;spacings
) ;foundry
) ;constraintGroups
```

See Mapping of the viaSpacing Table Definition.

DFII on CDB ASCII Technology File Definition:

Rule name: `viaSpacing`

```
physicalRules(
   tableSpacingRules(
   ( "viaSpacing"  "via1"
      ( "adjacentVias"   ( g_numCut{3|4})    nil )
      (
      (g_numAdjacentVia   ">="   )   g_viaCutSpacing
   );tableSpacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinAdjacentViaSpacing`
Value type: `oaIntValue`: spacing
Parameters: `oacDistanceConstraintParamType`,
`oacNumCutsConstraintParamType`,
`oaBoolean oacCenterToCenterConstraintParamType`,
`oaIntValue oacConnectivityTypeConstraintParamType`,
`oaBoolean oacExceptSamePGNetConstraintParamType`

LEF Syntax:

```
SPACING cutSpacing [CENTERTOCENTER] [SAMENET|SAMEMETAL|SAMEVIA]

ADJACENTCUTS {2|3|4} [EXACTALIGNED exactAlignedCut] WITHIN cutWithin

[EXCEPTSAMEPGNET] [CUTCLASS className]
```

Mapping of the viaSpacing Table Definition

In CDB, the `viaSpacing` rule can be defined using a table value. In OpenAccess the `viaSpacing` constraint is defined using a scalar value not a table value.

When mapping this rule to the Virtuoso environment, a table is converted to multiple scalar

`viaSpacing` constraints which will contain one value (spacing) and two parameters (number of adjacent cuts and distance).

CDB

```
( "viaSpacing"    "Cont"
   ("adjacentVias" nil nil)
   (
     3    0.4 4 0.5
   )
)
```

Virtuoso Studio design environment

```
( viaSpacing  "Cont"    (3 0.4 0.4) )
( viaSpacing  "Cont"    (4 0.4 0.5) )
```

Note: If the CDB table is a 1D table, then `spacing` is mapped to `distance`.

Minimum Via Spacing Constraint

This constraint specifies the via cut spacing for cuts on the same net or for cuts on the same metal. Optional parameters can be used to determine the following:

- Whether the cut spacing is to be measured edge-to-edge or center-to-center
- Whether the constraint applies to shapes having any connectivity, to shapes having only same-net connectivity, or only to cut shapes on the same metal
- Whether the cuts must have an area larger than a specified value before the spacing applies

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minViaSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ( minViaSpacing tx_cutLayer ['centerToCenter] ['sameNet|'sameMetal]
        ['area g_area] g_spacing )
   ) ;spacings
) ;foundry
) ;constraintGroups


constraintGroups(
("foundry"s
   spacingTables(
      ( minViaSpacing tx_cutLayer
      ( ("width" nil nil ["width" nil nil]) ['centerToCenter]
          ['sameNet|'sameMetal] ['area g_area] [g_default] )
       (g_table)
       )
   ) ;spacingTables
) ;foundry
) ;constraintGroups
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinViaSpacing`
Value type: `oaIntValue`,
`oaInt1DTblValue` (width),
`oaInt2DTblValue` (width, width): the spacing
Parameters: `oaBoolean oacCenterToCenterConstraintParamType`,
`oaIntValue oacConnectivityTypeConstraintParamType`,
`oaIntValue oacAreaConstraintParamType`

LEF Syntax:

```
SPACING cutSpacing [CENTERTOCENTER] [SAMENET | SAMEMETAL] AREA cutArea
```

Merge Allowed Spacing Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `maxFilling`

```
constraintGroups(
("foundry"
   spacings(
      ("maxFilling" tx_layer g_value)
      ...
   );spacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `maxFilling`

```
physicalRules(
   spacingRules(
      (maxFilling tx_layer g_value)
   );spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMergeSpaceAllowed`
Value type: `oaIntValue`

Minimum Width Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minWidth`

```
constraintGroups(
("foundry"
   spacings(
      ("minWidth" tx_layer g_width)
      ...
   );spacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `minWidth`

```
physicalRules(
   spacingRules(
      ("minWidth" tx_layer g_width)
   );spacingRules
);physicalRules
```

The rule `defaultMinWidth` is mapped according to the following convention:

- If the technology file contains `defaultMinWidth` but not `minWidth`, `defaultMinWidth` maps to the `minWidth` constraint with this value to both the `foundry` and the `LEFDefaultRouteSpec` constraint groups.
- If both rules are present and `minWidth` < `defaultMinWidth`, then the `defaultMinWidth` rule is mapped to a `minWidth` constraint in the `LEFDefaultRouteSpec` constraint group and the `minWidth` constraint is moved to the `foundry` constraint group.
- If both rules are present and `minWidth` > `defaultMinWidth`, the `minWidth` constraint moves to the `LEFDefaultRouteSpec` group and the `defaultMinWidth` constraint maps to a `minWidth` constraint in the `foundry` constraint group.

The rule `defaultWidth` is mapped to the `minWidth` constraint in the `LEFDefaultRouteSpec` constraint group.

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinWidth`
Value type: `oaIntValue`

Minimum Protrusion Width Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `protrusionWidth`

```
constraintGroups(
("foundry"
   spacings(
      ("protrusionWidth" tx_layer ( g_length g_width g_protrusionWidth )
      )

...
   );spacings
);foundry
);constraintGroups
```

where:

g_protrusionWidth is the minimum width, in user units, of the protrusion.

g_length is the length, in user units, that the protrusion length must be less than for the constraint to apply.

g_width is the minimum width of the wire to which the protrusion connects for the constraint to apply.

DFII on CDB ASCII Technology File Definition:

Rule name: `protrusionWidth`

```
physicalRules(
   spacingRules(
      ("protrusionWidth" tx_layer (g_length g_width g_value1) )
   );spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinProtrusionWidth` with 2 width parameters.
Value type: `oaIntValue`

Maximum Width Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `maxWidth`

```
constraintGroups(
("foundry"
   spacings(
      ("maxWidth" tx_layer g_width)
      ...
   );spacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `maxWidth`

```
physicalRules(
   spacingRules(
      ("maxWidth" tx_layer g_width)
   );spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMaxWidth`
Value type: `oaIntValue`

Minimum Length Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minLength`

```
constraintGroups(
("foundry"
   spacings(
      ("minLength" tx_layer g_length)
      ...
   );spacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `minLength`

```
physicalRules(
   spacingRules(
      ("minLength" tx_layer g_length)
   );spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinLength`
Value type: `oaIntValue`

Maximum Length Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `maxLength`

```
constraintGroups(
("foundry"
   spacings(
      ("maxLength" tx_layer g_length)
      ...
   );spacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `maxLength`

```
physicalRules(
   spacingRules(
      ("maxLength" tx_layer g_length)
   );spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMaxLength`
Value type: `oaIntValue`

Minimum Number of Cuts Constraint

This constraint specifies the minimum number of cuts a via can have when it connects either two wide wire shapes or a wide shape to a pin. The constraint applies when the width of the wire or pin shape being connected is more than the width specified by this constraint. An optional parameter can be used to specify the distance at which cut shapes can be from each other.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minNumCut`

```
constraintGroups(
("foundry"
   spacingTables(
      ( minNumCut tx_cutLayer
         (( "width" nil nil ) ['distanceWithin g_within] [ g_defaultValue ])
         ( g_table )
      )
   ) ;spacingTables
) ;foundry
) ;constraintGroups
```

When a LEF file is read in, if this constraint is set for any routing layers then the information is moved to the cut layers above and below the routing layer.

In case of conflicts, warnings are issued. LEF In will only store the value on the cut layers. LEF Out reconstructs the ABOVE/BELOW LEF data by looking at the values stored on the CUT layers.

DFII on CDB ASCII Technology File Definition:

Rule name: `minNumCut`

```
physicalRules(
   spacingRules(
      ("minNumCut" (g_numberOfCuts "width" g_width) )
   );spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint` (cut layer only)
Constraint type: `oacMinNumCut`
Value type: `oaInt1DTblValue` (width), number of cuts
Parameters: `oacDistanceWithinConstraintParamType`

LEF Syntax:

```
[MINIMUMCUT numCuts WIDTH width [WITHIN cutDistance]
   [FROMABOVE|FROMBELOW]
   [LENGTH wireLength WITHIN wireDistance];]...
```

Minimum Number of Cuts on Protrusion Constraint

This constraint specifies the number of via cuts required to connect a thin wire to a wide wire or pin. The constraint applies when the wide wire width and length exceed those specified. An optional parameter specifies the distance at which cut shapes can be from each other.

Tip

To allow for the specification of multiple `oacMinProtrusionNumCut` constraints, all of which must be specified, use the AND constraint group semantic. For more information about the AND semantic, see Constraint Group Semantics on page 105.

Note: In the example below, since both the `oacMinProtrusionNumCut` constraints are to be satisfied, they cannot be added directly to the `foundry` constraint group.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minProtrusionNumCut`

```
constraintGroups(
( "foundry"
      memberConstraintGroups(
        "protrusionGroup"
      ) ;memberConstraintGroups
   ) ;spacings
) ;foundry
( "protrusionGroup" nil nil 'and
      spacings(
      ( minProtrusionNumCut tx_cutLayer 'distance g_distance 'length g_length
        'width g_width ['distanceWithin g_distanceWithin] g_numCuts )
      ( minProtrusionNumCut tx_cutLayer 'distance g_distance 'length g_length
        'width g_width ['distanceWithin g_distanceWithin] g_numCuts )
     ) ;spacings
) ;protrusionGroup
) ;constraintGroups
```

OpenAccess Constraint:

Class: `oaLayerConstraint` (cut layer only)
Constraint type: `oacMinProtrusionNumCut` with 3 possible parameters, distance, length and width
Value type: `oaIntValue`, number of cuts (width, length, distance)
Parameters: `oacDistanceWithinConstraintParamType cutDistance`,
`oacWidthConstraintParamType`,
`oacLengthConstraintParamType`,
`oacDistanceConstraintParamType`

LEF Syntax:

```
[MINIMUMCUT numCuts WIDTH width [WITHIN cutDistance]
   [FROMABOVE|FROMBELOW]
   [LENGTH wireLength WITHIN wireDistance];]...
```

Allowed Shape Angles Constraint

This constraint specifies the allowed default geometry orientations (all angles, 45/135 degree and orthogonal, orthogonal only). The default orientation applies to all the layers in the technology.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `allowedShapeAngles`

```
constraintGroups(
("foundry"
   allowedShapeAngles( ["anyAngle"|"orthogonal"|"diagonal"]
         ( tx_layer "anyAngle" )
         ( tx_layer "orthogonal" )
         ( tx_layer "diagonal" )
   ) ;allowedShapeAngles
) ;foundry
) ;constraintGroups
```

OpenAccess Constraint:

Class: `oaSimpleConstraint`, `oaLayerConstraint`
Constraint type: `oacShapeAngle`, `oacLayerShapeAngle`
Value type: `oaShapeAngleType`,
`oaIntValue`: all, orthogonal, diagonal

Diagonal Shapes Allowed Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `diagonalShapesAllowed`

```
constraintGroups(
("foundry"
   spacings(
      ("diagonalShapesAllowed" tx_layer g_value)
      ...
   ) ;spacings
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Constraint name: `diagonalShapesAllowed`

```
physicalRules(
   spacingRules(
      ("diagonalShapesAllowed" tx_layer g_value)
   ) ;spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacDiagonalShapesAllowed`
Value type: `oaBoolValue`

Maximum Tap Spacing Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `maxTapSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ("maxTapSpacing" tx_layer g_spacing)
      ...
   ) ;spacings
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

```
physicalRules(
   spacingRules(
      ("maxTapSpacing" tx_layer g_spacing)
   ) ;spacingRules
) ;physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMaxTapSpacing`
Value type: `oaIntValue`

Minimum Notch Spacing

When the length of a notch is less than the specified distance, the spacing within the notch should satisfy the minimum notch spacing.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minNotchSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ( minNotchSpacing tx_layer 'notchLength g_notchLength g_spacing )
   ) ;spacings
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinNotchSpacing`
Value type: `oaIntValue`, minimum notch spacing
Parameters: `oacNotchLength`

LEF Syntax:

```
[SPACING minNotchSpacing NOTCHLENGTH minNotchLength ;]
```

Minimum End of Notch Spacing

The end of a notch is the bottom end of a U-shaped notch. This constraint specifies the minimum spacing between the notch and the shapes (on the same layer) overlapping within the extent of the notch. This constraint applies if all of the following hold true:

- The width of the end of notch is less than the specified width
- The notch length is greater than or equal to the minimum notch length
- The notch spacing is less than or equal to the minimum notch spacing

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minEndOfNotchSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ( minEndOfNotchSpacing tx_layer 'notchWidth g_notchWidth
        'notchLength g_notchLength 'notchSpacing g_notchSpacing
         g_endOfNotchSpacing )
   ) ;spacings
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinEndOfNotchSpacing`
Value type: `oaIntValue`, minimum end of notch spacing
Parameters: `oacNotchWidthConstraintParamType`,
`oacNotchLengthConstraintParamType`,
`oacNotchSpacingConstraintParamType`

LEF Syntax:

```
[[SPACING minNotchSpacing [ENDOFNOTCHWIDTH endOfNotchWidth NOTCHSPACING
minNotchSpacing NOTCHLENGTH minNotchLength] ;]
```

Minimum Wire Extension Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minWireExtension`

```
constraintGroups(
("LEFDefaultRouteSpec"
   spacings(
      ("minWireExtension" tx_layer g_spacing [g_distance])
   ) ;spacings
) ;LEFDefaultRouteSpec
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

N/A

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinWireExtension`
Value type: `oaIntValue`

Minimum Boundary Extension Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minPRBoundaryExtension`

```
constraintGroups(
("foundry"
   spacings(
      ("minPRBoundaryExtension" tx_layer g_extension)
      ...
   ) ;spacings
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `minPRBoundaryExtension` and `minExtension` (with the LPP `prBoundary`).

```
physicalRules(
   spacingRules(
      ("minPRBoundaryExtension" tx_layer g_extension)
   ) ;spacingRules
) ;physicalRules


physicalRules(
   spacingRules(
      ("minExtension" "prBoundary" [tx_layerPurpose] g_extension)
   ) ;spacingRules
) ;physicalRules
```

Note: The two layer rule `minExtension` on LPP `prBoundary` `boundary` is mapped to the single layer constraint `minPRBoundaryExtension`. For example:

```
( minExtension("prBoundary" "boundary") "nwell" 0.4 )
```

maps to:

```
( "minPRBoundaryExtension" "nwell" 0.4 )
```

For more information, see Mapping Shape Based Boundaries on page 363.

For information about mapping the two layer `minExtension` rule, see Minimum Overlap Constraint on page 285.

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinimumBoundaryExtension`
Value type: `oaIntValue`

Keep prBoundary Shared Edges Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `keepPRBoundarySharedEdges`

```
constraintGroups(
("foundry"
   spacings(
      ( keepPRBoundarySharedEdges tx_layer )
   );spacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacKeepAlignedShapeAndBoundary`
Value type: `oaBooleanValue`
Objects: `oaPRBoundary.`

Minimum Boundary Interior Halo Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minPRBoundaryInteriorHalo`

```
constraintGroups(
("foundry"
   spacings(
      ("minPRBoundaryInteriorHalo" tx_layer g_distance ['coincidentAllowed])
      ...
   ) ;spacings
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `minPRBoundaryInteriorHalo`

```
physicalRules(
   spacingRules(
      ("minPRBoundaryInteriorHalo" tx_layer g_extension)
   ) ;spacingRules
) ;physicalRules
```

```
physicalRules(
   spacingRules(
      ("minEnclosure" "prBoundary" [tx_layerPurpose] g_extension)
   ) ;spacingRules
) ;physicalRules
```

Note: The two layer rule `minEnclosure` on LPP `prBoundary boundary` is mapped to the single layer constraint `minPRBoundaryInteriorHalo`. For example:

```
( minEnclosure("prBoundary" "boundary")  "metal1" 0.3)
```

maps to:

```
( "minPRBoundaryInteriorHalo" "metal1" 0.3 )
```

For more information, see Mapping Shape Based Boundaries on page 363.

For information about mapping the two layer `minEnclosure` rule, see Minimum Extension Constraint.

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinBoundaryInteriorHalo`
Value type: `oaIntValue`

Maximum Diagonal Edge Length Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `maxDiagonalEdgeLength`

```
constraintGroups(
("foundry"
   spacings(
      ( maxDiagonalEdgeLength tx_layer g_length )
   ) ;spacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMaxDiagonalEdgeLength`
Value type: `oaIntValue`

Redundant Via Setback Constraint

A redundant via is inserted in order to increase the successful connection rate of the metal shapes when there is the possibility of via failure. When placing the via down on the larger metal shape, vias conform to the minimum enclosure rule. In case of redundant vias, larger enclosure values are required for such vias when both the vias are on the same edge.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `redundantViaSetback`

```
constraintGroups(
("foundry"
   spacingTables(
      ( "redundantViaSetback" tx_layer1 tx_layer2
      (( "width" nil nil ))
      ( g_table )
      )
   ) ;spacingTables
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerPairConstraint` (not symmetric)
Constraint type: `oacMinRedundantViaSetback`
Value type: `oaInt1DTblValue`(width)

Maximum Routing Distance Constraint

Maximum allowed routing distance is used to restrict how much routing can be done on a particular layer and is applied on a per net basis. Usually used for poly routing.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `maximumRoutingDistance`

```
constraintGroups(
("LEFDefaultRouteSpec"
   interconnect(
      ( maximumRoutingDistance tx_layer g_distance
   ) ;interconnect
) ;LEFDefaultRouteSpec
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMaxRoutingDistance`
Value type: `oaIntValue`
Objects: `oacScalarNetType`, `oacBusNetBitType`

Minimum Parallel Via Spacing Constraint

This constraint supports 32 nm design rules. The cutClass constraint must be defined before this constraint. For more information, see `minParallelViaSpacing` in *Virtuoso Technology Data ASCII Files Reference*.

This constraint is similar to Minimum Parallel Via Clearance Constraint.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minParallelViaSpacing`

```
constraintGroups(
("foundry"
   spacings(
        ( minParallelViaSpacing tx_cutLayer
         ['centerToCenter] ['exceptSameNet|'exceptSameMetal|'exceptSameVia]
         g_spacing)
   ) ;spacings
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinParallelViaSpacing`
Value type: `oaIntValue`: spacing
Parameters:

- `oaIntValue enum`

  `cdcExceptConnectivityTypeConstraintParamType`: except same net, except same metal, except same via:
  `cdcSameNetConnectivityType`
  `cdcContiguousShapesConnectivityType (default)`
  `cdcSameViaConnectivityType`

- `oaBooleanValue`

  `oacCenterToCenterConstraintParamType`: `'centerToCenter`

LEF Syntax:

```
SPACING cutSpacing [CENTERTOCENTER] PARALLELOVERLAP
[EXCEPTSAMENET|EXCEPTSAMEMETAL|EXCEPTSAMEVIA]
```

Minimum Parallel Within Via Spacing Constraint

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

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minParallelWithinViaSpacing`

```
constraintGroups(
("foundry"
   spacings(
        ( minParallelWithinViaSpacing tx_cutLayer
         ['centerToCenter] ['exceptSameNet]
         'cutDistance g_distance g_spacing)
   ) ;spacings
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Constraint type: `oaLayerConstraint`
Constraint name: `cdcMinParallelWithinViaSpacing`
Value type: `oaIntValue`: spacing
Parameters:

- `oaIntValue`

  `oacCutDistanceConstraintParamType`: parallel within

- `oaIntValue enum`

  `cdcExceptConnectivityTypeConstraintParamType`: except same net, except same metal, except same via. Only same net applies: `cdcSameNetConnectivityType`

- `oaBooleanValue`

  `oacCenterToCenterConstraintParamType`

LEF Syntax:

```
SPACING cutSpacing [CENTERTOCENTER] PARALLELWITHIN within [EXCEPTSAMENET];
```

Minimum Parallel Span Spacing Constraint

For more information, see minParallelSpanSpacing in *Virtuoso Technology Data ASCII Files Reference*.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minParallelSpanSpacing`

```
constraintGroups(
("foundry"
   spacingTables(
        ( minParallelSpanSpacing tx_layer1
        ( ( "span" nil nil "span" nil nil ) 'length g_length [g_defaultValue] )
        (g_table)
        )
   ) ;spacingTables
) ;foundry
);constraintGroups
```

OpenAccess Constraint:

Constraint type: `oaLayerConstraint`
Constraint name: `cdcMinParallelSpanSpacing`
Value: `oaInt2DTblValue`: (span length, span length) minimum spacing
Parameters:

- `oaIntValue`

  `oacLengthConstraintParamType`: parallel run length

LEF Syntax:

```
SPACINGTABLE PARALLELSPANLENGTH PRL runLength
  {SPANLENGTH spanLength {spacing}...};
```

Minimum Same Metal Shared Edge Via Spacing Constraint

For more information, see `minSameMetalSharedEdgeViaSpacing` in *Virtuoso Technology Data ASCII Files Reference*. The `cutClass` constraint must be defined before this constraint.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minSameMetalSharedEdgeViaSpacing`

```
constraintGroups(
("foundry"
   spacings(
        ( minSameMetalSharedEdgeViaSpacing tx_cutLayer
          'within g_within ['above]
          ['cutClass g_width|(g_width g_length)|t_name]
          ['exceptTwoEdges] ['numCuts g_cuts] g_spacing)
   ) ;spacings
) ;foundry
);constraintGroups
```

where the cut class width and length will be mapped to the previously defined cut class constraint whose name parameter matches the cut class width and length if defined, or will match the cut class width and length directly.

OpenAccess Constraint:

Constraint type: `oaLayerConstraint`
Constraint name: cdc`MinSameMetalSharedEdgeViaSpacing`
Value: `oaIntValue`: spacing
Parameters:

- `oaDualIntValue`

  `cdcCutClassConstraintParamType`: cut class (width, length)

- `oaIntValue`

  `oacParallelEdgeWithinConstraintParamType`: parallel within
  `cdcExceptSameViaCountConstraintParamType`: except same via num cuts

- `oaBooleanValue`

  `cdcExceptTwoEdgesConstraintParamType`
  `cdcAboveOnlyConstraintParamType`

LEF Syntax:

```
SPACING cutSpacing SAMEMETALSHAREDEDGE parWithin [ABOVE] [CUTCLASS class]
  [EXCEPTTWOEDGES] [EXCEPTSAMEVIA numCut];
```

Minimum End of Line Spacing Constraint

This constraint supports 32 nm design rules. For more information, see minEndOfLineSpacing in *Virtuoso Technology Data ASCII Files Reference*.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minEndOfLineSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ( minEndOfLineSpacing tx_layer 'width g_width ['oppWidth g_oppWidth]
      'distance g_distance
      ['endToEndSpace g_endToEndSpace ['otherEndWidth g_otherEndWidth]]
      ['maxLength g_maxLength | 'minLength g_minLength ['twoSides]]
      ['equalRectWidth]
      ['paraEdgeWithin g_parWithin 'paraEdgeSpace 'g_parSpace
      ['paraEdgeCount g_edgeCount] ['subtractWidth g_subtractWidth]
      ['paraMinLength g_parMinLength]]
      ['encloseDistance g_encloseDist 'cutToMetalSpace
      g_cutToMetalSpace ['above|'below] ['allCuts]] g_spacing
   ) ;spacings
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinEndOfLineSpacing`
Value type: `oaIntValue`
Parameters:

- `oaIntValue:`

  `oacWidthConstraintParamType`: `'width` (end of line width)
  `oacDistanceConstraintParamType`: `'distance` (within)
  `oacParallelEdgeSpaceConstraintParamType`: `'paraEdgeSpace` (parallel edge spacing)
  `oacParallelEdgeWithinConstraintParamType`: `'paraEdgeWithin` (parallel edge within)
  `oacCountConstraintParamType`: `'paraEdgeCount` (number of parallel edges)
  `cdcMinOppositeWidthConstraintParamType`: `'oppWidth` (opposite width)
  `cdcEndToEndSpacingConstraintParamType`: `'endToEndSpace` (end to end spacing)
  `cdcOtherEndWidthConstraintParamType`: `'otherEndWidth` (other end width)
  `oacMaxLengthConstraintParamType`: `'maxLength` (maximum length)
  `cdcMinLengthConstraintParamType`: `'minLength` (minimum length)
  `cdcParallelEdgeMinLengthConstraintParamType`: `'paraMinLength` (minimum length of end of line for parallel edge check)

  `cdcEnclosedDistanceConstraintParamType`: `'encloseDistance` (enclosure distance for cut shape)

  `cdcCutToMetalSpacingConstraintParamType`: `'cutToMetalSpace` (cut to metal spacing)

- `oaBooleanValue:`

  `cdcTwoSidesConstraintParamType`: `'twoSides`
  `cdcEqualRectWidthConstraintParamType`: `'equalRectWidth`
  `cdcSubtractEndOfLineWidthConstraintParamType`: `'subtractWidth`
  `cdcAllCutsConstraintParamType`: `'allCuts`

- `oaEnum:`

  `cdcEnclosedCutConstraintParamType: above, below, any` (default is any), `cdcAboveEnclosedCutType`, `cdcBelowEnclosedCutType` and `cdcAnyEnclosedCutType`: `'above` and `'below`

LEF Syntax:

```
SPACING eolSpace ENDOFLINE eolWidth [OPPOSITEWIDTH oppositeWidth] WITHIN eolWithin
  [ENDTOEND endToEndSpace [OTHERENDWITDTH otherEndWidth]]
  [MAXLENGTH maxLength | MINLENGTH minLength [TWOSIDES]]
  [EQUALRECTWIDTH]
  [PARALLELEDGE [SUBTRACTEOLWIDTH] parSpace WITHIN parWithin
    [MINLENGTH minLength] [TWOEDGES]]
  [ENCLOSECUT [BELOW | ABOVE] encloseDist CUTSPACING cutToMetalSpace
  [ALLCUTS]];
```

Minimum End of Line Perpendicular Spacing Constraint

This constraint supports 32 nm design rules. For more information, see `minEndOfLinePerpSpacing` in *Virtuoso Technology Data ASCII Files Reference*.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minEndOfLinePerpSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ( minEndOfLinePerpSpacing tx_layer 'width g_width 'perpWidth g_perpWidth
       g_spacing )
   ) ;spacings
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Constraint type: `oaLayerConstraint`
Constraint name: `cdcMinEndOfLinePerpSpacing`
Value: `oaIntValue`: end of line spacing
Parameters: `oaIntValue`:

`oacWidthConstraintParamType`: end of line width
`cdcEndOfLinePerpWidthConstraintParamType`: perpendicular width

LEF Syntax:

```
SPACING eolSpace EOLPERPENdICULAR eolWidth perWidth;
```

Minimum Large Via Array Spacing Constraint

This constraint defines the spacing between square arrays of vias, depending on the size of arrays. This constraint supports 32 nm design rules. The cutClass constraint must be defined before this constraint. For more information, see `minLargeViaArraySpacing` in *Virtuoso Technology Data ASCII Files Reference*.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minLargeViaArraySpacing`

```
constraintGroups(
("foundry"
   spacingTables(
      ( minLargeViaArraySpacing tx_cutLayer
      (( "numCuts" nil nil ) ['cutClass g_width | (g_width g_length) | t_name]
      ['paraOverlap] [g_defaultValue])
      (g_table))
   ) ;spacingTables
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinLargeViaArraySpacing`
Value type: `oaInt1DTblValue`: `numCuts`, spacing
Parameters:

- `oaDualIntValue`

  `cdcCutClassConstraintParamType`: `'cutClass` (width, length)

- `oaBooleanValue`

  `cdcParallelOverlapConstraintParamType`: `'paraOverlap`

LEF Syntax:

```
ARRAYSPACING [CUTCLASS className] [PARALLELOVERLAP] [LONGARRAY]
  [WIDTH viaWidth] CUTSPACING cutSpacing
  {ARRAYCUTS arrayCuts SPACING arraySpacing}...;
```

Minimum Large Via Array Cut Spacing Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minLargeViaArrayCutSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ( minLargeViaArrayCutSpacing tx_layer g_spacing ['numCuts numCuts]
   ) ;spacings
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerConstraint` (cut layer only)
Constraint type: `oacMinLargeViaArrayCutSpacing`
Value type: `oaIntValue cutSpacing`
Parameters: `oacMinNumCutsConstraintParamType`, `oaIntValue`, `numCuts`

Maximum Number of Edges with Minimum Edge Length Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `maxNumMinEdges`

```
constraintGroups(
("foundry"
   spacings(
      ( maxNumMinEdges tx_layer g_length)
   ) ;spacings
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinEdgeMaxCount`
Value type: `oaIntValue`
Parameters: `oacLengthConstraintParamType`, `oaIntValue`, `edgeLength`

Minimum Distance Between Adjacent Sets of Edges with Minimum Edge Length Constraint

This constraint defines the minimum distance between two consecutive edges that can satisfy the minimum edge length constraint. This constraint supports 32 nm design rules. For more information, see `minEdgeAdjacentDistance` in *Virtuoso Technology Data ASCII Files Reference*.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minEdgeAdjacentDistance`

```
constraintGroups(
("foundry"
   spacings(
      ( minEdgeAdjacentDistance tx_layer 'length g_length 'edgeCount g_edgeCount
      ['exceptSameCorner] g_distance )
   ) ;spacings
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinEdgeAdjacentDistance`
Value type: `oaIntValue`: minimum distance
Parameters:

- `oaIntValue`

  `oacMaxLengthConstraintParamType`: `'length` (maximum value of minimum edge length)
  `oacCountConstraintParamType`: `'edgeCount`

- `oaBooleanValue`

  `cdcExceptSameCornerConstraintParamType`: `'exceptSameCorner`

LEF Syntax:

```
MINSTEP minStepLength
  [MAXEDGES maxEdges MINBETWEENLENGTH minBetweenLength [EXCEPTSAMECORNERS]]
```

Minimum Length of Sets of Edges Adjacent to a Short Edge Constraint

This constraint defines the minimum length of edges adjacent to an edge that is shorter than a specified length. This constraint supports 32 nm design rules. For more information, see `minEdgeAdjacentLength` in *Virtuoso Technology Data ASCII Files Reference*.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minEdgeAdjacentLength`

```
constraintGroups(
("foundry"
   spacings(
      ( minEdgeAdjacentLength tx_layer 'maxLength g_maxLength
         g_maxEdges ['convexCorner] g_length )
      ( minEdgeAdjacentLength tx_layer 'maxLength g_maxLength
         g_maxEdges (g_length1 g_length2) )
   ) ;spacings
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint Type: `oacMinEdgeAdjacentLength`
Value Types: `oaIntValue`: the minimum length of the adjacent edges
`oaDualIntValue`: the minimum length for each of the two adjacent edges
Parameters:

- `oaIntValue`

  `oacMaxLengthConstraintParamType`: '`maxLength` (maximum length of a short edge)

- `oaBooleanValue`

  `cdcConvexCornerConstraintParamType`: '`convexCorner` (default is false). This applies only to `oaIntValue` and not to `oaDualIntValue`.

LEF Syntax:

```
MINSTEP minStepLength
  [MAXEDGES maxEdges
  [MINADJACENTLENGTH minAdjLength [CONVEXCORNER | minAdjLength2]];
```

Minimum Rectangle Area Constraint

This constraint is similar to the Minimum Area Constraint that applies to all shapes.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minRectArea`

```
constraintGroups(
("foundry"
   spacings(
      ( minRectArea tx_layer g_area )
   ) ;spacings
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint Type: `oacMinRectArea`
Value Type: `oaIntValue`: the minimum area in database units squared

Minimum Corner to Corner Distance Constraint

This constraint specifies the distance between inside concave corners of a shape. For more information, see minCornerToCornerDistance in *Virtuoso Technology Data ASCII Files Reference*.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minCornerToCornerDistance`

```
constraintGroups(
("foundry"
   spacings(
      ( minCornerToCornerDistance tx_layer1 g_width )
   ) ;spacings
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Constraint Type: `oaLayerConstraint`
Constraint Name: `cdcMinCornerToCornerDistance`
Value Type: `oaIntValue`: minimum distance
Parameters: None

LEF Syntax:

NA

Allowed Spacing Range Constraint

This constraint defines the spacings or spacing ranges required between gates below a given minimum width. The allowable spacings can depend on the width of either one shape or both. An optional width parameter can be specified such that the constraint applies only when both widths of the adjacent shapes are less than the given value. This constraint is defined at a process level and does not apply to a particular portion of a design.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `allowedSpacingRanges`

```
constraintGroups(
("foundry"
   spacings(
      ( allowedSpacingRanges tx_layer 'width g_width (g_ranges) )
   ) ;spacings
) ;foundry
);constraintGroups


constraintGroups(
("foundry"
   spacingTables(
      ( allowedSpacingRanges tx_layer
        (("width nil nil ["width nil nil]) ['width g_width] [g_default]) (g_table)
      )
   ) ;spacingTables
) ;foundry
);constraintGroups
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint Type: `oacAllowedSpacingRange`
Value Types: `oaIntRangeArrayValue`: array of `oaIntRange` values
`oaIntRangeArray1DTblValue` (width)
`oaIntRangeArray2DTblValue` (width, width): set of legal spacing/ranges
Parameter: `oacWidthConstraintParamType`

Taper Halo Constraint

This constraint specifies the offset around a pin shape where constraints such as minimum width and minimum spacing should be applied. This constraint can be applied to both design and technology databases.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `taperHalo`

```
constraintGroups(
("foundry"
   spacings(
      ( taperHalo g_offset )
   ) ;spacings
) ;foundry
);constraintGroups
```

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint Type: `oacTaperHalo`
Value Types: `oaIntValue`: offset around pin shape
Objects: `oaInstTerm`, `oaScalarTerm`, `oaBusTermBit`, `oaScalarNet`, `oaBusNetBit`, `oaPin`, `oaAppObject`

Corner Constraints

Minimum Outside Corner Edge Length Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minOutsideCornerEdgeLength`

```
constraintGroups(
("foundry"
   spacings(
      ("minOutsideCornerEdgeLength" tx_layer g_lengthDefinition)
      ...
   );spacings
);foundry
);constraintGroups
```

g_lengthDefinition can specify a length value only:

g_length

or

specify a length value and a length sum value:

(g_numCuts g_distance g_space)

where:

g_lengthSum is the maximum sum of the lengths of the consecutive edges, in user units.

g_length is the minimum length of each edge, in user units.

DFII on CDB ASCII Technology File Definition:

In OpenAccess 2.2, three minimum edge length constraints have been added to represent different types of edges: `minOutsideCornerEdgeLength`, `minInsideCornerEdgeLength`, and `minStepEdgeLength`.

Rule names: `minConvexEdgeLength`

```
physicalRules(
   spacingRules(
      ("minConvexEdgeLength" tx_layer (g_lengthSum g_length) )
   );spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinConvexEdgeLength`
Value type: `oaIntValue` with optional `oaConstraintParamArray` to store `oacLengthSumConstraintParamType`

Minimum Inside Corner Edge Length Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minInsideCornerEdgeLength`

```
constraintGroups(
("foundry"
   spacings(
      ("minInsideCornerEdgeLength" tx_layer (g_lengthSum g_length)
     ...
   );spacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

In OpenAccess 2.2, three minimum edge length constraints have been added to represent different types of edges: `minOutsideCornerEdgeLength`, `minInsideCornerEdgeLength`, and `minStepEdgeLength`.

Rule names: `minConcaveEdgeLength`

```
physicalRules(
   spacingRules(
      ("minConcaveEdgeLength" tx_layer (g_lengthSum g_length) )
   );spacingRules
);physicalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacMinConcaveEdgeLength`
Value type: `oaIntValue` with optional `oaConstraintParamArray` to store `oacLengthSumConstraintParamType`

Minimum Inside Corner Overlap Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minInsideCornerOverlap`

```
constraintGroups(
("foundry"
   orderedSpacings(
      ( minInsideCornerOverlap tx_layer1 tx_layer2 g_overlap)
   ) ;orderedSpacings
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerPairConstraint` (not symmetric)
Constraint type: `oacMinConcaveCornerOverlap`
Value type: `oaIntValue`

Minimum Inside Corner Extension Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minInsideCornerExtension`

```
constraintGroups(
("foundry"
   orderedSpacings(
      (minInsideCornerExtension tx_layer1 tx_layer2 g_extension )
   ) ;orderedSpacings
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerPairConstraint` (not symmetric)
Constraint type: `oacMinConcaveCornerExtension`
Value type: `oaIntValue`

Single Layer Routing Grids

Previously, routing grid information was specified as a layer attribute. LEF 5.6 supports a different pitch for the x and y directions. Since different pitch values can be applied to the same physical layer, routing grid information is specified as a constraint that allows different routers to use different routing grids.

Note: When using LEF In, the routing grid pitch and offset is mapped to the `LEFDefaultRouteSpec`. For information about mapping prRule routing grid information, see prRule Routing Grids. For information about preferred routing direction, see Preferred Routing Direction.

The routing grid constraints include:

- Horizontal Routing Grid Pitch Constraint
- Horizontal Routing Grid Offset Constraint
- Vertical Routing Grid Pitch Constraint
- Vertical Routing Grid Offset Constraint
- Left Diagonal Routing Grid Pitch Constraint
- Left Diagonal Routing Grid Offset Constraint
- Right Diagonal Routing Grid Pitch Constraint
- Right Diagonal Routing Grid Offset Constraint

The following default routing grid constraints are used if a layer-specific routing pitch or offset is not defined. These default routing grid constraints allow the specification of routing grids that apply to all routing layers:

- Default Horizontal Routing Grid Pitch Constraint
- Default Horizontal Routing Grid Offset Constraint
- Default Vertical Routing Grid Pitch Constraint
- Default Vertical Routing Grid Offset Constraint
- Default Left Diagonal Routing Grid Pitch Constraint
- Default Left Diagonal Routing Grid Offset Constraint
- Default Right Diagonal Routing Grid Pitch Constraint
- Default Right Diagonal Routing Grid Offset Constraint

Horizontal Routing Grid Pitch Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `horizontalPitch`

```
constraintGroups(
("foundry"
   routingGrids(
      ( horizontalPitch tx_layer g_pitch )
      ...
   ) ;routingGrids
) ;foundry
) ;constraintGroups
```

See Mapping of Pitch and Offset Using LEF In.

DFII on CDB ASCII Technology File Definition:

Attribute name: `horizontal`, `horizontalRouteGridPitch`

```
layerRules(
   layerRoutingGrids(
      (tx_layer "horizontal" g_pitch g_offset)
   ) ;layerRoutingGrids
) ;layerRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacHorizontalRouteGridPitch`
Value type: `oaIntValue`

Horizontal Routing Grid Offset Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `horizontalOffset`

```
constraintGroups(
("foundry"
   routingGrids(
      ( horizontalOffset tx_layer g_offset )
      ...
   ) ;routingGrids
) ;foundry
) ;constraintGroups
```

See Mapping of Pitch and Offset Using LEF In.

DFII on CDB ASCII Technology File Definition:

Attribute name: `offset`, `horizontal`, `horizontalRouteGridPitchOffset`

```
layerRules(
   layerRoutingGrids(
      (tx_layer "horizontal" g_pitch g_offset)
   ) ;layerRoutingGrids
) ;layerRules
```

`offset` is mapped to two constraints, `horizontalOffset` and `verticalOffset`.

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacHorizontalRouteGridOffset`
Value type: `oaIntValue`

Vertical Routing Grid Pitch Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `verticalPitch`

```
constraintGroups(
("foundry"
   routingGrids(
      ( verticalPitch tx_layer g_pitch )
      ...
   ) ;routingGrids
) ;foundry
) ;constraintGroups
```

See Mapping of Pitch and Offset Using LEF In.

DFII on CDB ASCII Technology File Definition:

Attribute name: `vertical`, `verticalRouteGridPitch`

```
layerRules(
   layerRoutingGrids(
      (tx_layer "vertical" g_pitch g_offset)
   ) ;layerRoutingGrids
) ;layerRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacVerticalRouteGridPitch`
Value type: `oaIntValue`

Vertical Routing Grid Offset Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `verticalOffset`

```
constraintGroups(
("foundry"
   routingGrids(
      ( verticalOffset tx_layer g_offset )
      ...
   ) ;routingGrids
) ; foundry
) ;constraintGroups
```

See Mapping of Pitch and Offset Using LEF In.

DFII on CDB ASCII Technology File Definition:

Attribute name: `offset`, `vertical`, `verticalRouteGridPitchOffset`

```
layerRules(
   layerRoutingGrids(
      (tx_layer "vertical" g_pitch g_offset)
   ) ;layerRoutingGrids
) ;layerRules
```

`offset` is mapped to two constraints, `horizontalOffset` and `verticalOffset`

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacVerticalRouteGridOffset`
Value type: `oaIntValue`

Left Diagonal Routing Grid Pitch Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `leftDiagPitch`

```
constraintGroups(
("foundry"
   routingGrids(
      ( leftDiagPitch tx_layer g_pitch )
      ...
   ) ;routingGrids
) ;foundry
) ;constraintGroups
```

See Mapping of Pitch and Offset Using LEF In.

DFII on CDB ASCII Technology File Definition:

N/A

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oac135RouteGridPitch`
Value type: `oaIntValue`

Left Diagonal Routing Grid Offset Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `leftDiagOffset`

```
constraintGroups(
("foundry"
   routingGrids(
      ( leftDiagOffset tx_layer g_offset )
      ...
   ) ;routingGrids
) ;foundry
) ;constraintGroups
```

See Mapping of Pitch and Offset Using LEF In.

DFII on CDB ASCII Technology File Definition:

N/A

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oac135RouteGridOffset`
Value type: `oaIntValue`

Right Diagonal Routing Grid Pitch Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `rightDiagPitch`

```
constraintGroups(
("foundry"
   routingGrids(
      ( rightDiagPitch tx_layer g_pitch )
      ...
   ) ;routingGrids
) ;foundry
) ;constraintGroups
```

See Mapping of Pitch and Offset Using LEF In.

DFII on CDB ASCII Technology File Definition:

N/A

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oac45RouteGridPitch`
Value type: `oaIntValue`

Right Diagonal Routing Grid Offset Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `rightDiagOffset`

```
constraintGroups(
("foundry"
   routingGrids(
      ( rightDiagOffset tx_layer g_offset )
      ...
   ) ;routingGrids
) ;foundry
) ;constraintGroups
```

See Mapping of Pitch and Offset Using LEF In.

DFII on CDB ASCII Technology File Definition:

N/A

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oac45RouteGridOffset`
Value type: `oaIntValue`

Default Horizontal Routing Grid Pitch Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `horizontalPitch`

```
constraintGroups(
("foundry"
   routingGrids(
      ( horizontalPitch g_pitch )
      ...
   ) ;routingGrids
) ;foundry
) ;constraintGroups
```

See Mapping of Pitch and Offset Using LEF In.

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacDefaultHorizontalRouteGridPitch`
Value type: `oaIntValue`

Default Horizontal Routing Grid Offset Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `horizontalOffset`

```
constraintGroups(
("foundry"
   routingGrids(
      ( horizontalOffset g_offset )
      ...
   ) ;routingGrids
) ;foundry
) ;constraintGroups
```

See Mapping of Pitch and Offset Using LEF In.

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacDefaultHorizontalRouteGridPOffset`
Value type: `oaIntValue`

Default Vertical Routing Grid Pitch Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `verticalPitch`

```
constraintGroups(
("foundry"
   routingGrids(
      ( verticalPitch g_pitch )
      ...
   ) ;routingGrids
) ;foundry
) ;constraintGroups
```

See Mapping of Pitch and Offset Using LEF In.

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacDefaultVerticalRouteGridPitch`
Value type: `oaIntValue`

Default Vertical Routing Grid Offset Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `verticalOffset`

```
constraintGroups(
("foundry"
   routingGrids(
      ( verticalOffset g_offset )
      ...
   ) ;routingGrids
) ; foundry
) ;constraintGroups
```

See Mapping of Pitch and Offset Using LEF In.

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacDefaultVerticalRouteGridOffset`
Value type: `oaIntValue`

Default Left Diagonal Routing Grid Pitch Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `leftDiagPitch`

```
constraintGroups(
("foundry"
   routingGrids(
      ( leftDiagPitch g_pitch )
      ...
   ) ;routingGrids
) ;foundry
) ;constraintGroups
```

See Mapping of Pitch and Offset Using LEF In.

DFII on CDB ASCII Technology File Definition:

N/A

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacDefault135RouteGridPitch`
Value type: `oaIntValue`

Default Left Diagonal Routing Grid Offset Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `leftDiagOffset`

```
constraintGroups(
("foundry"
   routingGrids(
      ( leftDiagOffset g_offset)
      ...
   ) ;routingGrids
) ;foundry
) ;constraintGroups
```

See Mapping of Pitch and Offset Using LEF In.

DFII on CDB ASCII Technology File Definition:

N/A

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacDefault135RouteGridOffset`
Value type: `oaIntValue`

Default Right Diagonal Routing Grid Pitch Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `rightDiagPitch`

```
constraintGroups(
("foundry"
   routingGrids(
      ( rightDiagPitch g_pitch )
      ...
   ) ;routingGrids
) ;foundry
) ;constraintGroups
```

See Mapping of Pitch and Offset Using LEF In.

DFII on CDB ASCII Technology File Definition:

N/A

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacDefault45RouteGridPitch`
Value type: `oaIntValue`

Default Right Diagonal Routing Grid Offset Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `rightDiagOffset`

```
constraintGroups(
("foundry"
   routingGrids(
      ( rightDiagOffset g_offset )
      ...
   ) ;routingGrids
) ;foundry
) ;constraintGroups
```

See Mapping of Pitch and Offset Using LEF In.

DFII on CDB ASCII Technology File Definition:

N/A

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacDefault45RouteGridOffset`
Value type: `oaIntValue`

Placement Grid Constraints

Horizontal Placement Grid Pitch Constraint

This constraint describes the space between each horizontal placement grid line.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `horizontalPitch`

```
constraintGroups(
("foundry"
   placementGrid(
      ( horizontalPitch g_pitch )
   ) ;placementGrid
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacHorizontalPlacementGridPitch`
Value type: `oaIntValue`
Objects: `oaBlock`

Horizontal Placement Grid Offset Constraint

This constraint describes the offset from the origin for the horizontal placement grid.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `horizontalOffset`

```
constraintGroups(
("foundry"
   placementGrid(
      ( horizontalOffset g_offset)
   ) ;placementGrid
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacHorizontalPlacementGridOffset`
Value type: `oaIntValue`
Objects: `oaBlock`

Vertical Placement Grid Pitch Constraint

This constraint describes the space between each vertical placement grid line.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `verticalPitch`

```
constraintGroups(
("foundry"
   placementGrid(
      ( verticalPitch g_pitch )
   ) ;placementGrid
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacVerticalPlacementGridPitch`
Value type: `oaIntValue`
Objects: `oaBlock`

Vertical Placement Grid Offset Constraint

This constraint describes the offset from the origin for the vertical placement grid.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `verticalOffset`

```
constraintGroups(
("foundry"
   placementGrid(
      ( verticalOffset g_offset)
   ) ;placementGrid
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacVerticalPlacementGridOffset`
Value type: `oaIntValue`
Objects: `oaBlock`

Antenna Models

The antenna model constraints specifies the antenna ratios for one oxide type corresponding to a specific thickness of the gate oxide.

There are two types of antenna calculation, per layer and cumulative. OpenAccess only supports either non-side or side antenna ratios, they cannot be specified simultaneously.

The antenna area factor is not stored as a separate value in OpenAccess, it is used to divide the other ratios. In OpenAccess cumulative antenna ratios defined for one layer will be propagated to other layers in the techfile.

When the technology file is updated or when using LEF In, the side value will override the non-side value if both are defined.

The antenna area factor is not stored as a separate value in OpenAccess, it is used to divide the other ratios. In OpenAccess cumulative antenna ratios defined for one layer will be propagated to other layers in the technology file.

In the following CDB example there are two types of gate ratios on a per layer and a cumulative basis and side and non-side:

```
electricalRules(
characterizationRules(
; (antennaRuleName layerName
; ------------------ ----------------------
; l_areaRatio l_sideAreaRatio
; ------------------ ----------------------
; l_diffAreaRatio l_diffSideAreaRatio
; ------------------ ----------------------
; l_cumAreaRatio l_cumDiffAreaRatio
; ------------------ ----------------------
; l_cumSideAreaRatio l_cumDiffSideAreaRatio
; ------------------ ----------------------
; l_areaFactor l_sideAreaFactor
; ------------------ ----------------------
( defaultAntennaRule metal1
20. 30.
((20. 2.) (30. 3.)) ((20. 2.) (30. 3.))
40. ((20. 2.) (30. 3.))
50. ((20. 2.) (30. 3.))
1.0 (0.5 t)
)
( secondAntennaRule metal1
20. 30.
((20. 2.) (30. 3.)) ((20. 2.) (30. 3.))
40. ((20. 2.) (30. 3.))
50. ((20. 2.) (30. 3.))
1.0 (0.5 t)
)
) ;characterizationRules
) ;electricalRules
```

```
gate : l_areaRatio, l_sideAreaRatio
cumGate : l_cumAreaRatio, l_cumSideAreaRatio
```

This is similarly for the diode ratios:

```
diode : l_diffAreaRatio, l_diffSideAreaRatio
cumDiode : l_cumDiffAreaRatio, l_cumDiffSideAreaRatio
```

For the area factors, given by `l_areaFactor` and `l_sideAreaFactor`, you can specify either `d_value` or `(g_value [t | nil])`. If `t` is specified then the area factor given by `g_value` only applies to diode antenna ratios. If `nil` or `d_value` is specified then area factor is applied to both gates and diodes antenna ratios. The default area factor value in both cases is `1.0`. The entries for diodes in the previous rules can be written in a list format to specify a `PWL` value `(g_area g_value)`.

In the Virtuoso Studio design environment, the `cumGate` and `cumDiode` entries map to a simple constraint. If the layer is metal, then `oacCumMetalAntenna` will be used. If the layer is via, then `oacCumViaAntenna` will be used.

If one of the values for `l_cumSideAreaRatio` or `l_cumDiffSideAreaRatio` has been specified then this value is used in preference to the non-side value and `isSide` is set to false. The gate and diode as defined above map to the `oacAntenna` layer constraint.

Note: The non-side value (`l_areaRatio` or `l_diffAreaRatio`) is used in preference to the side value to set the boolean for `isSide` to be false.

If the `areaFactor` value is not equal to `1.0`, then the antenna values in the technology file are divided by the appropriate `areaFactor`, paying attention to which gate or diode ratios the `areaFactor` applies.

Antenna Oxide1 Model Constraint

.

Virtuoso Studio design environment ASCII Technology File Definition:

```
antennaModels(
    ( "default"
       antenna(
         ( areaRatio lt_layer g_value )
         (diffAreaRatio  lt_layer g_value | ((g_area g_value)  [b_isSide] )
       ); antenna
cumulativeMetalAntenna(
       ( areaRatio   g_value [b_isSide] )
       (diffAreaRatiolt_layer g_value | ((g_area g_value)  [b_isSide] )
   ); cumulativeMetalAntenna
   cumulativeViaAntenna(
       ( areaRatio  g_value [b_isSide] )
       (diffAreaRatio  lt_layer g_value | ((g_area g_value)  [b_isSide] )
   ); cumulativeViaAntenna
) ;default
) ;antennaModels
```

DFII on CDB ASCII Technology File Definition:

Rule name: `defaultAntennaRule`

```
electricalRules(
characterizationRules(
   (defaultAntennaRule tx_layer
      (l_areaRatio l_sideAreaRatio
      l_diffAreaRatio l_diffSideAreaRatio
      l_cumAreaRatio l_cumDiffAreaRatio
      l_cumSideAreaRatio l_cumDiffSideAreaRatio
      l_areaFactor l_sideAreaFactor)
   ) ;defaultAntennaRule
) ;characterizationRules
) ;electricalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacAntenna`

Class: `oaSimpleConstraint:`
Constraint type: `oacCumMetalAntenna`

Class: `oaSimpleConstraint:`
Constraint type: `oacCumViaAntenna`

Value type: `oaAntennaRatioArrayValue`
on Tech DB with:

```
oaAntennaRatioValue
oaBoolean isSide
oaFloat gateRatio
oa1DLookupTbl(oaInt8, oaFloat) diodeRatio
```

Antenna Oxide2 Model Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

```
antennaModels(
    ( "second"
       antenna(
         ( areaRatio lt_layer g_value )
         (diffAreaRatio  lt_layer g_value | ((g_area g_value)  [b_isSide] )
       ); antenna
cumulativeMetalAntenna(
       ( areaRatio   g_value [b_isSide] )
       (diffAreaRatiolt_layer g_value | ((g_area g_value)  [b_isSide] )
   ); cumulativeMetalAntenna
   cumulativeViaAntenna(
       ( areaRatio  g_value [b_isSide] )
       (diffAreaRatio  lt_layer g_value | ((g_area g_value)  [b_isSide] )
   ); cumulativeViaAntenna
) ;second
) ;antennaModels
```

DFII on CDB ASCII Technology File Definition:

Rule name: `secondAntennaRule`

```
electricalRules(
characterizationRules(
   (secondAntennaRule tx_layer
   (l_areaRatio l_sideAreaRatio
   l_diffAreaRatio l_diffSideAreaRatio
   l_cumAreaRatio l_cumDiffAreaRatio
   l_cumSideAreaRatio l_cumDiffSideAreaRatio
   l_areaFactor l_sideAreaFactor)
   ) ;secondAntennaRule
) ;characterizationRules
) ;electricalRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacAntenna`

Class: `oaSimpleConstraint:`
Constraint type: `oacCumMetalAntenna`

Class: `oaSimpleConstraint::`
Constraint type: `oacCumViaAntenna`

Value type: `oaAntennaRatioArrayValue`
on Tech DB with:

```
oaAntennaRatioValue
oaBoolean isSide
oaFloat gateRatio
oa1DLookupTbl(oaInt8, oaFloat) diodeRatio
```

Antenna Oxide3 Model Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

```
antennaModels(
    ( "third"
       antenna(
         ( areaRatio lt_layer g_value )
         (diffAreaRatio  lt_layer g_value | ((g_area g_value)  [b_isSide] )
       ); antenna
cumulativeMetalAntenna(
       ( areaRatio   g_value [b_isSide] )
       (diffAreaRatiolt_layer g_value | ((g_area g_value)  [b_isSide] )
   ); cumulativeMetalAntenna
   cumulativeViaAntenna(
       ( areaRatio  g_value [b_isSide] )
       (diffAreaRatio  lt_layer g_value | ((g_area g_value)  [b_isSide] )
   ); cumulativeViaAntenna
) ;third
) ;antennaModels
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacAntenna`

Class: `oaSimpleConstraint:`
Constraint type: `oacCumMetalAntenna`

Class: `oaSimpleConstraint:`
Constraint type: `oacCumViaAntenna`

Value type: `oaAntennaRatioArrayValue`
on Tech DB with:
`oaAntennaRatioValue`
`oaBoolean isSide`
`oaFloat gateRatio`
`oa1DLookupTbl(oaInt8, oaFloat) diodeRatio`

Antenna Oxide4 Model Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

```
antennaModels(
    ( "fourth"
       antenna(
         ( areaRatio lt_layer g_value )
         (diffAreaRatio  lt_layer g_value | ((g_area g_value)  [b_isSide] )
       ); antenna
cumulativeMetalAntenna(
       ( areaRatio   g_value [b_isSide] )
       (diffAreaRatiolt_layer g_value | ((g_area g_value)  [b_isSide] )
   ); cumulativeMetalAntenna
   cumulativeViaAntenna(
       ( areaRatio  g_value [b_isSide] )
       (diffAreaRatio  lt_layer g_value | ((g_area g_value)  [b_isSide] )
   ); cumulativeViaAntenna
) ;fourth
) ;antennaModels
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
Constraint type: `oacAntenna`

Class: `oaSimpleConstraint:`
Constraint type: `oacCumMetalAntenna`

Class: `oaSimpleConstraint:`
Constraint type: `oacCumViaAntenna`

Value type: `oaAntennaRatioArrayValue`
on Tech DB with:
`oaAntennaRatioValue`
`oaBoolean isSide`
`oaFloat gateRatio`
`oa1DLookupTbl(oaInt8, oaFloat) diodeRatio`

Two Layer Constraints

Minimum Clearance Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ("minSpacing" tx_layer1 tx_layer2 g_distance)
      ...
   ) ;spacings
) ;foundry
) ;constraintGroups
```

Minimum spacing can be based on the width of two shapes. The minimum spacing between two shapes on the same layer where the width of one of the shapes is width1 and the width of the other shape is width2. Constraint name: `minSpacing` Value: `width`

```
constraintGroups(
("foundry"
   spacingTables(
      (( "minSpacing" tx_layer1 tx_layer2
      (("width" nil nil ["width"|"length" nil nil]))
      ( g_table )
      )
   ) ;spacingTables
) ;foundry
);constraintGroups
```

```
constraintGroups(
("foundry"
   spacingTables(
      ( "minSpacing" tx_layer1 tx_layer2
      (( "width" nil nil nil nil ))
      ( g_table )
   )
   ) ;spacingTables
);foundry
);constraintGroups
```

OpenAccess Constraint:

Class: `oaLayerPairConstraint`
Constraint type: `oacMinClearance` (symmetric)
Parameter: `oacDistanceMeasureTypeConstraintParamType`
Value type: `oaIntValue`

Maximum Spacing Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `maxSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ("maxSpacing" tx_layer1 tx_layer2 g_distance)

   ...
   ) ;spacings
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `maxSpacing`

```
physicalRules(
   spacingRules(
      ("maxSpacing" tx_layer1 tx_layer2 g_distance)
   ) ;spacingRules
) ;physicalRules
```

OpenAccess Constraint:

Class: `oaLayerPairConstraint`
Constraint type: `oacMaxClearance` (symmetric)
Value type: `oaIntValue`

Minimum Clearance Constraint (sameNet)

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minSameNetSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ("minSameNetSpacing" tx_layer1 tx_layer2 g_distance)
      ...
   ) ;spacings
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule names: `minSameNetSpacing` and `sameNet`

```
physicalRules(
   spacingRules(
      ("minSameNetSpacing" tx_layer1 tx_layer2 g_distance)
   ) ;spacingRules
) ;physicalRules
```

The LEF `SAMENET` statement was previously mapped to the `sameNet` rule in the `spacingRules` section.

```
physicalRules(
   spacingRules(
      ("sameNet" tx_layer1 tx_layer2 g_distance)
   ) ;spacingRules
) ;physicalRules
```

OpenAccess Constraint:

Class: `oaLayerPairConstraint`
Constraint type: `oacMinSameNetClearance` (symmetric)
Value type: `oaIntValue`

Minimum Extension Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minExtension`

```
constraintGroups(
("foundry"
   orderedSpacings(
      ("minExtension" tx_layer1 tx_layer2 g_spacing )
      ...
   ) ;orderedSpacings
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `minExtension`

```
physicalRules(
   orderedSpacingRules(
      ("minExtension" tx_layer1 tx_layer2 g_spacing)
   ); orderedSpacingRules
) ;physicalRules
```

If the CDB rule `minExtension` specifies the layer `prBoundary`, then the rule will map to Minimum Boundary Extension Constraint on page 217.

OpenAccess Constraint:

Class: `oaLayerPairConstraint`
Constraint type: `oacMinOverlap` (not symmetric)
`oaConstraintParam oacDistanceConstraintParamType`
Value type: `oaIntValue`

Minimum Opposite Extension Constraint

This constraint specifies the minimum distance a shape on one layer must extend past a shape on a second layer. The extension is measured from the outside edge of the shape on the second layer to the inside edge of the shape on the first layer. If there are more than one minimum opposite extension constraints specified for a given width, only one of the constraints is required to be satisfied.

This constraint supports 32 nm design rules. The cutClass constraint must be defined before this constraint. For more information, see `minOppExtension` in *Virtuoso Technology Data ASCII Files Reference*.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minOppExtension`

```
constraintGroups(
("foundry"
   orderedSpacings(
      ( minOppExtension tx_layer1 tx_layer2
        ['cutClass g_width | (g_width g_length) | t_name ['endSide]]
        ['cutDistance g_distance ['noSharedEdge | 'hasParaRunLength]]
        ['length g_length | 'extraCut | 'redundantCutWithin g_distance]
        (g_ext g_oppExt) )
   ) ;orderedSpacings
) ;foundry
) ;constraintGroups


constraintGroups(
("foundry"
   spacingTables(
      ( minOppExtension tx_layer1 tx_layer2
         ( ( "width" nil nil )
         ['cutClass g_width | (g_width g_length) | t_name ['endSide]]
         ['cutDistance g_distance ['noSharedEdge | 'hasParaRunLength]]
         ['length g_length | 'extraCut | 'redundantCutWithin g_distance]
         [g_default] )
         ( g_table )
      )
   ) ;spacingTables
) ;foundry
) ;constraintGroups
```

where a table row is of the format:

```
 ( g_width ( (g_ext g_oppExt)
```

or

```
  ( g_width ( (g_ext1 g_oppExt1) (g_ext2 g_oppExt2) ...) )
```

OpenAccess Constraint:

Class: `oaLayerArrayConstraint` (not symmetric)
Constraint type: `oacMinDualExtension`
Value type: `oacDualIntValue`, `oacDualInt1DTblValue` (width), `oacIntDualIntArrayTblValue` (width)

Parameters:

- `oaIntValue`

  `oacLengthConstraintParamType`: `'length` (minimum length)
  `oacCutDistanceConstraintParamType`: `'cutDistance` (for the cut within)
  `oacNumCutsConstraintParamType` with set value of 2
  `cdcRedundantCutDistanceConstraintParamType`: `'redundantCutWithin` (for redundant cut within)

- `oaBooleanValue`

  `oacNoSharedEdgeConstraintParamType`: `'noSharedEdge`. Default value is `false`.
  `cdcHasParallelRunLengthConstraintParamType`: `'hasParaRunLength` (mutually exclusive with no shared edge parameter). Default value is `false`.
  `cdcEndSideOverhangConstraintParamType`: `'endSide` (If true, the first value corresponds to the overhang of the end or short edge, and the second value corresponds to the overhang of the side or long edge). Default value is `false`. This applies only to rectangular cut class constraints.

- `oaDualIntValue`

  `cdcCutClassConstraintParamType`: `'cutClass`

LEF Syntax:

```
ENCLOSURE [CUTCALSS className] [ABOVE|BELOW] {overhang1 overhang2 | END overhang1
SIDE overhang2}
   [WIDTH minWidth [EXCEPTEXTRACUT cutWithin [PRL | NOSHAREDEDGE]]
   |LENGTH minLength
   |EXTRACUT
   |REDUNDANTCUT cutWithin];
```

Preferred Opposite Extension Constraint

The `oacMinDualExtension` constraint can be defined in a precedence constraint group, in which the constraints are ordered first by width and cut distance, then by width, and then by

length. The constraint definition is not symmetric implying that the minimum extension of layer1 past layer2 is not the same as the minimum extension of layer2 past layer1.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minOppExtension`

```
constraintGroups(
("foundry"
   orderedSpacings(
      ( minOppExtension tx_layer1 tx_layer2 ['length g_length] ['cutDistance
        g_distance] ['noSharedEdge] ( g_extension1 g_extension2 ) 'soft )
   ) ;orderedSpacings
) ;foundry
) ;constraintGroups

constraintGroups(
("foundry"
   spacingTables(
      ( minOppExtension tx_layer1 tx_layer2 'soft )
         ( ( "width" nil nil ) ['length g_length] ['cutDistance
             g_distance] ['noSharedEdge] [ g_defaultValue ] )
         ( g_table )
   ) ;spacingTables
) ;foundry
) ;constraintGroups
```

where a table row is of the format:

```
( g_width ( (g_extension1 g_oppExtension1) (g_extension2 g_oppExtension2) ...) )
```

OpenAccess Constraint:

Class: `oaLayerPairConstraint` (two layers)
Constraint type: `oacMinDualExtension` (not symmetric)
Value type: `oacDualIntValue`, pair of extension values
`oacDualInt1DTblValue` pair of extension values indexed by width
`oacIntDualIntArrayTblValue`

Parameters: `oacLengthConstraintParamType` for minimum length,
`oacCutDistanceConstraintParamType` for the cut within,
`oacNoSharedEdgeConstraintParamType` (boolean with default value `false`)

Optional `oacDistanceConstraintParamType` for the distance to neighboring shapes.

LEF Syntax:

```
[PREFERENCLOSURE [ABOVE|BELOW] overhang1 overhang2
   [WIDTH minWidth] ;]

PROPERTY LEF57 PREFERENCLOSURE
"PREFERENCLOSURE [ABOVE|BELOW] overhang1 overhang2
   [WIDTH minWidth] ;
```

Minimum Extension Edge Constraint

For more information, see `minExtensionEdge` in *Virtuoso Technology Data ASCII Files Reference*. The `cutClass` constraint must be defined before this constraint.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minExtensionEdge`

```
constraintGroups(
("foundry"
   spacings(
      ( minExtensionEdge tx_layer1 tx_layer2 'width g_width 'parallelLength
        g_parLength 'within g_within
        ['exceptExtraCut ['distanceWithin g_distance]]
        ['exceptTwoEdges]
        ['cutClass g_width|(g_width g_length)|t_name] g_extension )
   ) ;spacings
) ;foundry
) ;constraintGroups
```

where the cut class width and length will be mapped to the previously defined cut class constraint whose name parameter matches the cut class width and length if defined, or will use the cut class width and length directly.

OpenAccess Constraint:

Constraint type: `oaLayerPairConstraint` (metal layer, cut layer)
Constraint name: `cdcMinExtensionEdge`
Value type: `oacIntValue`: edge extension
Parameters:

- `oaDualIntValue`:

  `cdcCutClassConstraintParamType`: cut class defined by width and length

- `oaIntValue`:

  `oacWidthConstraintParamType`: width
  `oacParallelEdgeWithinConstraintParamType`: parWithin
  `cdcParallelEdgeLengthConstraintParamType`: parLength
  `oacDistanceWithinConstraintParamType`: cut within

- `oaBooleanValue`:

  `cdcExceptTwoEdgesConstraintParamType`
  `cdcExceptExtraCutConstraintParamType`

LEF Syntax:

```
ENCLOSUREEDGE [CUTCLASS className]
  [ABOVE|BELOW] overhang WIDTH minWidth
  PARALLEL parLength WITHIN parWithin
  [EXCEPTEXTRACUT [cutWithin]] [EXCEPTTWOEDGES];
```

Minimum Orthogonal Via Spacing Constraint

The minimum orthogonal spacing between a pair of cut shapes and any other cut shape is the spacing between the cuts and the spacing to any other via cut shape in an orthogonal direction from the imaginary line between the cut pair. A range of allowed spacings is defined by the distance between the cut pair and the result is the orthogonal spacing allowed to other cut shapes.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minOrthogonalViaSpacing`

```
constraintGroups(
("foundry"
   spacingTables(
      ( minOrthogonalViaSpacing tx_cutLayer
       (( "distance" nil nil) [g_default] )
       ( g_table )
      )
   ) ;spacingTables
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerConstraint` (cut layer only)
Constraint type: `oacMinOrthogonalViaSpacing`
Value type: `oacInt1DTblValue` (spacing between cut pair), orthogonal spacing

LEF Syntax:

```
[SPACINGTABLE ORTHOGONAL
   {WITHIN cutWithin SPACING orthoSpacing}...;}
```

Coincident Extension Allowed Constraint

This constraint specifies that abutted objects with co-incident edges are allowed. This constraint can be used in conjunction with any other constraint. By default, it is set to false.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: Parameter `['coincidentAllowed]`

`( t_constraint lx_layer1 [lx_layer2] g_value ['coincidentAllowed] )`

DFII on CDB ASCII Technology File Definition:

N/A

OpenAccess Constraint:

Class: `oaConstraintParamDef`

Parameter `oacCoincidentAllowedParamType` on any constraint.

List of Constraints that Support the Parameter `'coincidentAllowed`

OA Constraint	DFII Constraint
oacMinBoundaryExtension	minPRBoundaryExtension
oacMinWireExtension	minWireExtension
oacMinBoundaryInteriorHalo	minPRBoundaryInteriorHalo
	minPRBoundaryExteriorHalo
oacMinExtension	minEnclosure
oacMaxExtension	maxEnclosure
	maxExtension
oacMinDualExtension	minOppExtension
oacMinDiffPotentialSpacing	minDiffNetSpacing
oacMinSameNetSpacing	minSameNetSpacing
cdcMinCutClassSpacing	minCutClassSpacing
	minExtensionDistance

Minimum Overlap Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minOverlap`

```
constraintGroups(
("foundry"
   spacings(
      ("minOverlap" tx_layer1 tx_layer2 g_overlap)
      ...
   ) ;spacings
) ;foundry
) ;constraintGroups
```

Note: The OpenAccess recommended representation is to use the minimum area constraint on a derived layer produced by the “and” of the two original layers.

DFII on CDB ASCII Technology File Definition:

N/A

OpenAccess Constraint:

N/A

Minimum Parallel Via Clearance Constraint

The minimum parallel via clearance is the minimum separation required between the cut shapes on one layer with respect to the cut shapes with parallel edges on an adjacent cut layer. This constraint applies when vias overlap and do not share the same metal shape on the layer between them. This constraint is similar to Minimum Parallel Via Spacing Constraint that applies only to two cut layers.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minParallelViaSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ( minParallelViaSpacing tx_cutLayer1 tx_cutLayer2 tx_metalLayer g_spacing
       )
   ) ;spacings
) ;foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerArrayConstraint` (two cut layers and a metal layer between the two cut layer; ordered layer triple)
Constraint Type: `oacMinParallelViaClearance`
Value Type: `oaIntValue`

Via Stacking Rule Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `stackable`

```
layerRules(
   functions(
      ( tx_layer "cut" )
      ( tx_layer "li" )
   ); functions
);layerRules


constraintGroups(
("foundry"
   spacings(
      ("stackable" tx_layer1 tx_layer2 g_boolean)
      ...
   ) ;spacings
) ;foundry
) ;constraintGroups
```

Also see, prRule Stack Vias.

DFII on CDB ASCII Technology File Definition:

Rule name: `stackable`

```
physicalRules(
   spacingRules(
      ("stackable" tx_layer1 tx_layer2 g_boolean)
   )
)
```

OpenAccess Constraint:

Class: `oaLayerPairConstraint`
Constraint type: `oacViaStackingAllowed` (symmetric)
Value type: `oaBooleanValue`
Value type: function type `cut` or `li`

Minimum Enclosure Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minEnclosure`

```
constraintGroups(
("foundry"
   orderedSpacings(
      ("minEnclosure" tx_layer1 tx_layer2 g_distance)
      ...
   ) ;orderedSpacings
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

Rule name: `minEnclosure`

```
physicalRules(
   orderedSpacingRules(
      ("minEnclosure" tx_layer1 tx_layer2 g_boolean)
   ) ;orderedSpacingRules
) ;physicalRules
```

If the CDB rule `minEnclosure` specifies the layer `prBoundary`, then the rule will map to Minimum Boundary Interior Halo Constraint on page 219.

OpenAccess Constraint:

Class: `oaLayerPairConstraint`
Constraint type: `oacMinExtension` (not symmetric)
Value type: `oaIntValue`, `oatInt1DTblValue(width)`

Maximum Enclosure Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `maxEnclosure`

```
constraintGroups(
("foundry"
   orderSpacings(
      ("maxEnclosure" tx_layer1 tx_layer2 g_distance)
      ...
   ) ;orderSpacings
) ;foundry
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

```
physicalRules(
   orderedSpacingRules(
      ("maxExtension" tx_layer1 tx_layer2 g_boolean)
   ) ;orderedSpacingRules
) ;physicalRules
```

OpenAccess Constraint:

Class: `oaLayerPairConstraint`
Constraint type: `oacMaxExtension` (not symmetric)
Value type: `oaIntValue`, `oatInt1DTblValue(width)`

Minimum End of Line Extension Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minEndOfLineExtension`

```
constraintGroups(
("foundry"
   orderSpacings(
      ("minEndOfLineExtension" tx_layer1 tx_layer2 g_width g_within g_spacing
g_parallelEdgeWithin g_parallelEdgeSpace g_extension)
      ...
   ) ;orderSpacings
) ;foundry
) ;constraintGroups
```

OpenAccess Constraint:

Class: `oaLayerPairConstraint`
Constraint type: `oacMinEndOfLineExtension`
Value type: `oaIntValue`
Parameter: `oacWidthConstraintParamType` eolWidth,
`oacDistanceConstraintParamType` eolWithin,
`oacEndOfLineSpaceConstraintParamType` eolSpace,
`oacParallelEdgeWithinConstraintParamType` parEdgeWithin
`oacParallelEdgeSpaceConstraintType` parEdgeSpace and

Keep Shared Edges Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `keepSharedEdges`

```
constraintGroups(
("foundry"
   spacings(
      ( keepSharedEdges tx_layer1 tx_layer2 ['buttOnly|'coinOnly] )
      )
   );spacings
); foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerPairConstraint` (symmetric)
Constraint type: `oacKeepAlignedShapes`
Value type: `oaBooleanValue`
Parameter: `enum oaIntValue`, `oaButtOnly oaCoinOnly`, `oaBoth`

Allowed Clearance Range Constraint

This constraint defines the spacings or spacing ranges required between gates below a given minimum width and other shapes on the second layer. The allowable spacings can depend on the width of either one shape or both. An optional width parameter can be specified such that the constraint applies only when both widths of the adjacent shapes are less than the given value. This constraint is defined at a process level and does not apply to a particular portion of a design.

This constraint applies when the cumulative run length of between the gate shape and the facing poly shape is greater than or equal to 20 percent of the gate width.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `allowedSpacingRanges`

```
constraintGroups(
("foundry"
   spacings(
      ( allowedSpacingRanges tx_layer1 tx_layer2 'width g_width
        (g_ranges) )
   ) ;spacings
) ;foundry
);constraintGroups


constraintGroups(
("foundry"
   spacingTables(
      ( allowedSpacingRanges tx_layer1 tx_layer2
        (("width nil nil ["width nil nil]) ['width g_width] [g_default]) (g_table)
      )
   ) ;spacingTables
) ;foundry
);constraintGroups
```

OpenAccess Constraint:

Class: `oaLayerPairConstraint` (ordered layer pair)
Constraint Type: `oacAllowedClearanceRange`
Value Types: `oaIntRangeArrayValue`: array of `oaIntRange` values
`oaIntRangeArray1DTblValue` (width)
`oaIntRangeArray2DTblValue` (width, width): set of legal clearances/ranges
Parameter: `oacWidthConstraintParamType` width

Minimum Via Clearance Constraint

This constraint specifies the via cut clearance between cuts on different layers. Optional parameters can be used to determine the following:

- Whether the clearance is to be measured edge-to-edge or center-to-center
- Whether the constraint applies to shapes having any connectivity, to shapes having only same-net connectivity, or only to cut shapes on the same metal
- Whether the cuts with an area larger than a specified value must meet the specified clearance
- Whether same-net cuts on different layers can be stacked

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minViaSpacing`

```
constraintGroups(
("foundry"
   spacings(
      ( minViaSpacing tx_cutLayer1 tx_cutLayer2 ['centerToCenter]
        ['sameNet|'sameMetal] ['stack] ['area g_area] g_spacing )
   ) ;spacings
) ;foundry
) ;constraintGroups


constraintGroups(
("foundry"
   spacingTables(
      ( minViaSpacing tx_cutLayer1 tx_cutLayer2
      ( ("width" nil nil ["width" nil nil]) ['centerToCenter]
          ['sameNet|'sameMetal] ['stack] ['area g_area] [g_default] )
       (g_table)
       )
   ) ;spacingTables
) ;foundry
) ;constraintGroups
```

OpenAccess Constraint:

Class: `oaLayerPairConstraint` (unordered)
Constraint type: `oacMinViaClearance`
Value type: `oaIntValue`,
`oaInt1DTblValue` (width),
`oaInt2DTblValue` (width, width): the clearance
Parameters: `oaBoolean oacCenterToCenterConstraintParamType`,
`oaIntValue oacConnectivityTypeConstraintParamType`,
`oaIntValue oacAreaConstraintParamType`
`oaBoolean oacStackConstraintParamType`

LEF Syntax:

```
SPACING cutSpacing [CENTERTOCENTER] [SAMENET | SAMEMETAL] LAYER layerName [STACK]
```

Three Layers Spacing Constraints

Minimum Enclosure in Direction of Touching Layer Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minTouchingDirEnclosure`

```
constraintGroups(
("foundry"
   orderedSpacings(
      ( minTouchingDirEnclosure tx_layer1 tx_layer2 tx_layer3 g_enclosure
      ['manhattan])
   ) ;orderedSpacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerTripleConstraint` (not symmetric)
Constraint type: `oacMinTouchingDirectionExtension`
Value type: `oaIntValue`
Parameter: `oaConstraintParam`: `oaBooleanValue` `oaEuclidean`: default is true

Minimum Spacing in Direction of Touching Layer Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minTouchingDirSpacing`

```
constraintGroups(
("foundry"
   orderedSpacings(
      ( minTouchingDirSpacing tx_layer1 tx_layer2 tx_layer3 g_spacing
      ['manhattan])
      ( minTouchingDirSpacing tx_layer1 tx_layer2 tx_layer3 (g_spacing
       g_oppSpacing) ['manhattan])
   ) ;orderedSpacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerTripleConstraint` (not symmetric)

Constraint type: `oacMinTouchingDirectionClearance`

Value type: `oaIntValue`, `oaDualIntValue`

Minimum Spacing Over Layer Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `minSpacingOver`

```
constraintGroups(
("foundry"
   orderedSpacings(
      ( minSpacingOver tx_layer1 tx_layer2 tx_layer3 g_spacing )
   ) ;orderedSpacings
);foundry
);constraintGroups
```

DFII on CDB ASCII Technology File Definition:

NA

OpenAccess Constraint:

Class: `oaLayerTripleConstraint` (symmetric for layer1 and layer2 only)
Constraint type: `oacMinClearanceOverLayer`
Value type: `oaIntValue`

Dummy Poly Extension Constraint

In processes that require dummy poly shapes to exist on either side of a gate, both poly and gate shapes must be extended with the dummy poly shapes so that they overlap the adjacent active layer by a given maximum amount. This constraint specifies the required extension of the dummy poly shapes beyond the adjacent shapes on the active layer. This constraint does not apply if a gate is already joined to another gate above or below.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `dummyExtension`

```
constraintGroups(
("foundry"
   orderedSpacings(
      ( dummyExtension tx_layer1 tx_layer2 tx_layer3 'width g_width g_extension )
   ) ;orderedSpacings
);foundry
);constraintGroups
```

OpenAccess Constraint:

Class: `oaLayerArrayConstraint` (first layer is derived gate layer, second layer is dummy poly, and third layer is active layer)
Constraint type: `oacDummyPolyExtension`
Value type: `oaIntValue`, the extension
Parameters: `oacWidthConstraintParamType`, the maximum dummy poly width

LEF Syntax:

NA

Constraints Associated with the Technology Database

This section covers technology constraints which are associated with the technology database rather than applied to a specific layer.

Maximum Number of Via Stacking Constraint

A via is considered to be part of a stack if the cut shapes partially overlap other cut shapes in the via stack. This constraint specifies the maximum number of single or multiple cut vias that are allowed on top of one another in a continuous stack. The boolean parameter (`oacNoSingleCutViaConstraintParam`), if specified, ensures that no single cut via is allowed in the stack.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `viaStackingLimits`

```
constraintGroups(
("foundry"
   viaStackingLimits(
      ( g_stackLimits ['noSingleCut] [lx_bottomLayer lx_topLayer] )
   ) ;viaStackingLimits
) ;foundry
) ;constraintGroups
```

Also see, prRule Max Stack Vias.

DFII on CDB ASCII Technology File Definition:

Subclass: `viaStackingLimits`

```
physicalRules(
   viaStackLimits(
      (g_stackLimits [tx_bottomLayer tx_topLayer] )
   ) ;viaStackLimits
) ;physicalRules
```

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacViaStackLimit`
Value type: `oaIntValue`, number of stack vias
Parameters: `oacLowerLayerConstraintParamType`,
`oacUpperLayerConstraintParamType`,
`oacIntValue`, maximum number of vias in stack
`oaBoolean oacNoSingleCutViaConstraintParam`

LEF Syntax:

```
PROPERTY LEF57_MAXVIASTACK "MAXVIASTACK maxStack [NOSINGLE] [RANGE bottomLayer
topLayer];";
[MAXVIASTACK maxStack [NOSINGLE] [RANGE bottomLayer topLayer];]
```

Valid Routing Layers Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `validLayers`

```
constraintGroups(
("LEFDefaultRouteSpec"
   interconnect(
      ( validLayers l_layers)
   ) ;interconnect
) ;LEFDefaultRouteSpec
) ;constraintGroups
```

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacValidRoutingLayers`
Value type: `oaLayerArrayValue`

Valid Routing Vias Constraint

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `validVias`

```
constraintGroups(
( LEFDefaultRouteSpec
   interconnect(
      ( validVias (t_viaName/t_viaVariantName...) )
      ( validVias (
        (( "width" nil nil "width" nil nil) g_defaultValue )
        (
        (g_layer1Width g_layer2Width) (t_viaName/t_viaVariantName...)
         ...
        )
        )
   ) ;interconnect
) ;LEFDefaultRouteSpec
) ;constraintGroups
```

where `g_defaultValue` is *`t_viaName/t_viaVariantName`*

Note: The default value for the 2D table version is not an optional parameter.

DFII on CDB ASCII Technology File Definition:

N/A

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacValidRoutingVias`
Value type: `oaViaTopologyArrayValue`,
`oaViaTopology2DTblValue(layer1Width,layer2Width)`

Parameters: None

Objects: `oaRoute`, `oaScalarNet`, `oaBusNetBit`, `oaAppObject`

Global Distance Measure

This constraint applies to all constraints by default unless it has an explicit overriding value.

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint name: `distanceMeasure`

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

OpenAccess Constraint:

Constraint type: `oaClearanceMeasure`
Value type: `oaClearanceMeasureEnum` (default: Euclidian)
If `oacEuclidianClearanceMeasure`, distance is measured in Euclidian directions
If `oacMaxXYClearanceMeasure`, distance is measured in Manhattan directions

LEF Syntax:

```
[CLEARANCEMEASURE {MAXXY | EUCLIDIAN} ;]
```

Conversion of Technology File electricalRules Class

CDB	Virtuoso Studio design environment
electricalRules orderedCharacterizationRules characterizationRules tableCharacterizationRules	See links below.

Note: This class has been deprecated and will be removed in an upcoming release. Cadence recommends that you remove it from your technology files.

Changes:

Characterization Rules on page 304

Current Density Rules on page 304

Mapping User Defined Electrical Rules on page 304

Characterization Rules

Characterization rules are stored as properties of the physical layers. See Characterization Rules Stored as Layer Properties on page 132

Current Density Rules

The current density rules are now stored as layer attributes, not as technology constraints. See Current Density Rules on page 149

Mapping User Defined Electrical Rules

User defined electrical rules are mapped to the `electricalCharacterizations()` section within a constraint group.

OpenAccess rules/constraints are only applicable to layers, not layer purpose pairs. Electrical rules that are defined for LPPs are stored as user defined constraints and are only available to Virtuoso applications.

Note: Layers on purpose `drawing` are considered LPP rules. For example,

```
characterizationRules(
( edgeCap ("METAL1" "drawing" ) 9.8e-05 )
```

maps to

```
( "foundry"
   electrical(
      ( edgeCapacitance ("METAL1" "drawing") 9.8e-05)
   )
)
```

User Electrical Rules

```
constraintGroups(
( "foundry"
   ; electrical constraints
   ; user defined electrical constraints
   electrical(
   ;( constraint layer1 [layer2] value )
   ;( ---------- ------ -------- ----- )
   ( t_userConstraint tx_layer1 [tx_layer2] g_value )
   ) ;electrical
) ;foundry
) ;constraintGroups
```

User defined ordered electrical constraints are mapped to the `orderedElectricalCharacterizations()` section within a constraint group.

```
constraintGroups(
( "foundry"
   ; electrical constraints
   ; user defined electrical constraints
   orderedElectrical(
   ;( constraint layer1 [layer2] value )
   ;( ---------- ------ -------- ----- )
   ( t_userConstraint tx_layer1 [tx_layer2] g_value )
   ) ;orderedElectrical
) ;foundry
) ;constraintGroups
```

User defined table based rule are mapped to the `electricalCharacterizationsTable()` section within a constraint group.

```
constraintGroups(
( "foundry"
   ; electrical constraints
   ; user defined electrical constraints
   electricalTables(
   ;( constraint layer1 [layer2])
; (index1Definitions [index2Defintions]) [defaultValue]
; (table) )
;( ---------------------------------------------------)
( t_userConstraint tx_layer1 [tx_layer2]
( t_index1 nil nil [t_index2 nil nil]
( g_table )
); electrialTables
```

```
) ;foundry
) ;constraintGroups
```

Conversion of Technology File prRules Class

CDB	Virtuoso Studio design environment
prRules prMastersliceLayers prOverlapLayer prRoutingLayers prRoutingPitch prRoutingOffset prStackVias prMaxStackVias prViaTypes prViaRules prGenViaRules prNonDefaultRules	See links below.

CDB	Virtuoso Studio design environment

Note: This class has been deprecated and will be removed in an upcoming release. Cadence recommends that you remove it from your technology files.

Changes: prRule mapping is LEF5.5 compatible.

For information about prRule conversion error messages, see Appendix A, “prRules Conversion Messages.”

prRule Masterslice Layers on page 309

prRule Overlap Layers on page 310

prRule Routing Layers on page 311

prRule Routing Grids on page 312

prRule Stack Vias on page 314

prRule Max Stack Vias on page 315

prViaTypes on page 316

prViaTypes Default on page 317

prVia Rules on page 319

prGenViaRules on page 321

prNonDefaultRules on page 322

prRule Masterslice Layers

Virtuoso Studio design environment ASCII Technology File Definition:

```
layerRules(
   functions(
   ( tx_layer tx_layerFunction [g_maskNumber])
   ...
   ) ;functions
) ;layerRules
```

DFII on CDB ASCII Technology File Definition:

```
prRules(
   prMastersliceLayers(
      ( tx_layer ... )
      ...
   ) ;prMastersliceLayers
) ;prRules
```

OpenAccess Constraint:

`functions()`, `type "nDiff"`, `"pDiff"`, `"nWell"`, `"pWell"`, `"diff"` and `"poly"`

prRule Overlap Layers

Virtuoso Studio design environment ASCII Technology File Definition:

```
controls(
   techParams(
      ( LEFDEF_OVERLAP_LAYER_NAME t_name )
      ) ;techParams
) ;controls
```

DFII on CDB ASCII Technology File Definition:

```
prRules(
   prOverlapLayers((
   tx_layer ...
   ) ;prOverlapLayers
) ;prRules
```

OpenAccess Constraint:

Constraint type: oaLayer Property: `cLefOverlapLayer`

prRule Routing Layers

Virtuoso Studio design environment ASCII Technology File Definition:

```
layerRules(
   functions(
      ( tx_layer tx_layerFunction g_maskNumber)
      ...
   ) ;functions
routingDirections(
   ( tx_layer g_direction )
   ) ;routingDirections
) ;layerRules


constraintGroups(
("LEFDefaultRouteSpec"
   routingGrids(
      (tx_layer "tx_pitch" g_pitch [g_offset])
   ) ;routingGrids
      interconnect(
         ( "validLayers" tx_layerNames ... )
         ( "validVias" tx_viaNames ... )
      ) ;interconnect
) ;LEFDefaultRouteSpec
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

```
prRules(
   prRoutingLayers((
   tx_layer ...
   ) ;prRoutingLayers
) ;prRules
```

OpenAccess Constraint:

Layer functions type metal or poly in general.

Class: `oaSimpleConstraint`
Constraint type: `oacValidRoutingLayers`

oaConstraintGroup: `LEFDefaultRouteSpec`

Value type: `oaLayerArrayValue`

`oaPhysicalLayer` preferred routing direction.
Valid directions:
`vertical`, `horizontal`, `leftDiag`, `rightDiag`, `none`, or `notApplicable`.

prRule Routing Grids

In the Virtuoso environment, the routing grid information can be stored in the `LEFDefaultRouteSpec` constraint group. When querying the routing grid information, the order of precedence is the `LEFDefaultRouteSpec` and then the `foundry` constraint group.

Also see, Single Layer Routing Grids, and Preferred Routing Direction.

prRule Routing Pitch

Virtuoso Studio design environment ASCII Technology File Definition:

```
constraintGroups(
("LEFDefaultRouteSpec"
   routingGrids(
      (tx_layer "horizontalPitch" g_pitch [g_offset])
      (tx_layer "verticalPitch" g_pitch [g_offset])
      (tx_layer "leftDiagPitch" g_pitch [g_offset] )
      (tx_layer "rightDiagPitch" g_pitch [g_offset] )
   ) ;routingGrids
) ;LEFDefaultRouteSpec
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

```
prRules(
prRoutingLayers(
(metal1 "horizontal")
); prRoutingLayers

prRoutingPitch(
(metal1   2.4)
):prRoutingPitch

(metal2 "vertical")
); prRoutingLayers

prRoutingPitch(
(metal2 2.4)
):prRoutingPitch

); prRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
oaConstraintGroup: `oacHorizontalRouteGridPitch`, `oacVeriticalRouteGridPitch`

prRule Routing Grids Offset

Virtuoso Studio design environment ASCII Technology File Definition:

Constraint group: `LEFDefaultRouteSpec` (with `offset` value specified)

```
constraintGroups(
("LEFDefaultRouteSpec"
   routingGrids(
      (tx_layer "horizontalOffset" g_pitch [g_offset])
      (tx_layer "verticalOffset" g_pitch [g_offset])
      (tx_layer "leftDiagOffset" g_pitch [g_offset] )
      (tx_layer "rightDiagOffset" g_pitch [g_offset] )
   ) ;routingGrids
) ;LEFDefaultRouteSpec
) ;constraintGroups
```

DFII on CDB ASCII Technology File Definition:

```
prRules(
prRoutingLayers(
(metal1 "horizontal")
); prRoutingLayers

prRoutingPitch(
(metal1   2.4)
):prRoutingPitch

prRoutingOffset(
(metal1 0.0)
); prRoutingOffset

(metal2 "vertical")
); prRoutingLayers

prRoutingPitch(
(metal2 2.4)
):prRoutingPitch

prRoutingOffset(
(metal2 0.0)
); prRoutingOffset

); prRules
```

OpenAccess Constraint:

Class: `oaLayerConstraint`
oaConstraintGroup: `oacHorizontalRouteGridOffset`, `oacVeriticalRouteGridOffset`

prRule Stack Vias

Virtuoso Studio design environment ASCII Technology File Definition:

```
layerRules(
   functions(
      ( tx_layer "cut" )
      ( tx_layer "li" )
   ) ;functions
) ;layerRules

constraintGroups(
( "foundry" nil
   spacings(
      ( stackable tx_layer1 tx_layer2 g_boolean )
   ) ;spacings
) ;foundry
) ;constraintGroups
```

Also see, Via Stacking Rule Constraint.

DFII on CDB ASCII Technology File Definition:

```
prRules(
   prStackVias(
      ( lt_viaLayer lt_via2Layer )
   ) ;prStackVias
) ;prRules
```

OpenAccess Constraint:

Class: `oaLayerPairConstraint`
Constraint type: `oacViaStackingAllowed` (symmetric)
Value type: `oaBooleanValue`

Value type: function type `cut` or `li`

prRule Max Stack Vias

Virtuoso Studio design environment ASCII Technology File Definition:

```
constraintGroups(
("foundry"
   viaStackLimits(
      (g_stackLimits [tx_bottomLayer tx_topLayer] )
   ) ;viaStackLimits
) ;foundry
) ;constraintGroups
```

Also see, Maximum Number of Via Stacking Constraint.

DFII on CDB ASCII Technology File Definition:

```
prRules(
   prMaxStackVias(
      ( x_value [ ltx_bottomLayer ltx_topLayer ] )
      ...
   ) ;prMaxStackVias
) ;prRules
```

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacViaStackLimit` with 2 possible parameters, `oaConstraintParamArray` to store lower-Layer (`oacLayerConstraintType`) and upper-Layer (`oacLayerConstraintType`)
Value type: `oaIntValue`

prViaTypes

Virtuoso Studio design environment ASCII Technology File Definition:

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

Via is a via view created by LEF In.

Vias with via view names are not specified in the technology so an ASCII technology file cannot be used to create these views.

This type of via is used for power routing.

DFII on CDB ASCII Technology File Definition:

```
prRules(
   prViaTypes(
      ( l_cellView t_viaType )
   ...
   ) ;prViaTypes
) ;prRules

devices(
ruleContactDevice(
   ( g_deviceName
   ( tx_layer1   tx_purpose1   tx_rectangles ) | nil
   [( tx_viaLayer   tx_viaPurpose g_rectangles )]
   [( tx_layer2  tx_ purpose2   g_rectangles )]
) ;ruleContactDevice
) ;devices
```

OpenAccess Constraint:

Class: `oaViaDefs`
Constraint type: `oaCustomViaDefs`

An `oaCustomViaDef` which is not defined in an `oacValidRoutingVias` constraint.

prViaTypes Default

Virtuoso Studio design environment ASCII Technology File Definition:

For default via:

```
constraintGroups(
   ( "LEFDefaultRouteSpec" nil
      ( validVias tx_viaNames ... )
   ) ;LEFDefaultRouteSpec
) ;constraintGroups
viaDefs(
   customViaDefs(
       ( t_viaDefName t_libName t_cellName t_viewName
      tx_layer1 tx_layer2 [n_resistancePerCut] )
   ) ;customViaDefs
) ;viaDefs
```

The LEF In translator is customized for Virtuoso applications. Do not use the `lef2oa` translator for Virtuoso applications.

When using `lef2oa`, a default via in LEF is translate into a `customViaDef`. If you use `lefin` in CDB to translate a default via, it is translated to a `syContact`.

DFII on CDB ASCII Technology File Definition:

```
prRules(
   prViaTypes(
      ( l_cellView "default" )
   ...
   ) ;prViaTypes
) ;prRules


devices(
symContactDevice(
   ( g_deviceName
   ( tx_viaLayer1   tx_viaPurpose [implant1] )
   ( tx_viaLayer1   tx_viaPurpose [implant1] )
   g_width g_length [( row   column   xPitch   yPitch   xBias   yBias )]
   tx_encLayer1   etx_ncLayer2 g_legalRegion
) ;symContactDevice
) ;devices
```

Via is a symbolic view created by the technology file.

If the via is asymmetric then it is specified as:

```
devices(
cdsViaDevice(
(t_deviceName t_cutLayer t_cutPurpose
t_layer1 t_purpose1
t_layer2 t_purpose2
n_row n_column t_origin g_stackedVias
g_cutLayerW g_cutLayerL g_xCutSpacing g_yCutSpacing
g_layer1XDirOverride g_layer1YDirOverride
g_layer2XDirOverride g_layer2YDirOverride
t_layer1Dir t_layer2Dir
g_layer1XDefOverride g_layer1YDefOverride
g_layer2XDefOverride g_layer2YDefOverride
l_implantLayer1
l_implantLayer2
g_diffSpacing
t_abutClass
)
...
) ;cdsViaDevice
) ;devices
```

Via is a layout view created by the technology file.

OpenAccess Constraint:

Class: `oaViaDefs`
Constraint type: `oaCustomViaDefs`

Class: `oaSimpleConstraint`
Constraint type: `oacValidRoutingVias`

An `oaCustomViaDef` which is defined in an `oacValidRoutingVias` constraint in the `oaConstraintGroup`; `LEFDefaultRouteSpec`.

prVia Rules

Both `prViaRules` and `prGenViaRules` map to a `viaSpec` and the `viaSpec` is identified only by the layer pair. If `prViaRules` and `prGenViaRules` are defined on the same pair of layers, they will map to one `viaSpec`.

Note: `prViaRules` properties are not supported in OpenAccess. When loading a technology file containing prViaRules properties, a warning is issued stating that the properties will not be stored in the OpenAccess technology file.

Virtuoso Studio design environment ASCII Technology File Definition:

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

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

DFII on CDB ASCII Technology File Definition:

```
prRules(
   prViaTypes(
      ( l_cellView t_viaType )
   ...
   )
   prViaRules(
      ( t_ruleName l_viaNames
      lt_layer1 t_direction1 ( n_minWidth1 n_maxWidth1
         n_overhang1 n_metalOverhang1 )
      lt_layer2 t_direction2 ( g_minWidth2 g_maxWidth2
         g_overhang2 g_metalOverhang2 )
      [( properties )...] )
...
   ) ;prViaTypes
) ;prRules
```

Direction taken from layer direction.

`metalOverhang` not stored in OpenAccess.

OpenAccess Constraint:

Class: `oaViaSpecs`
Value type: `oaViaDefArrayValue, oaViaDef2DTblValue`

Class: `oaViaDefs`
Constraint type: `oaCustomViaDef`

prGenViaRules

Virtuoso Studio design environment ASCII Technology File Definition:

```
viaDefs(
standardViaDefs(
    ( t_viaDefName tx_layer1 tx_layer2
       ( t_cutLayer n_cutWidth n_cutHeight [n_resistancePerCut] )
       ( x_cutRows x_cutCol ( l_cutSpace ) )
        l_layer1Enc ) ( l_layer2Enc ) ( l_layer1Offset )
       ( l_layer2Offset ) ( l_origOffset )
       [tx_implant1 (l_implant1Enc) [tx_implant2 (l_implant2Enc]]
    )
) ;standardViaDefs
) ;viaDefs
viaSpecs(
    ( t_viaSpecName tx_layer1 tx_layer2 lt_viaDefNames
       (
       ([ n_layer1MinWidth n_layer1MaxWidth n_layer2MinWidth
n_layer2MaxWidth ]
       (lt_viaDefNames))
    )
) ;viaSpecs
```

DFII on CDB ASCII Technology File Definition:

```
prRules(
   prGenViaRules(
       ( t_ruleName lt_layer
       ( g_lowerPt g_upperPt g_xPitch g_yPitch g_resistance )
       ltx_layer1 t_dir1 | ( g_enc1Overhang1 g_enc1Overhang2 )
       (n_minWidth1 n_maxWidth1
          n_overhang1 n_metal1Overhang1 )
       ltx_layer2 t_dir2 [ ( g_enc2Overhang1 g_enc2Overhang2 )
       (n_minWidth2 n_maxWidth2
          n_overhang2 n_metal1Overhang2 )
       [( properties )...] )
       ...
   ) ;prGenViaRules
) ;prRules
```

OpenAccess Constraint:

Class: `oaViaSpec`
Value type: `oaViaDefArrayValue, oaViaDef2DTblValue`

Class: `oaViaDefs`
Constraint type: `oaStdViaDef`

prNonDefaultRules

Nondefault rules are used to specify non default vias for signal routing. This is similar to the via rule LEF construct used for power routing. In LEF 5.5 the non default rule specifies which vias should be used when the rule is satisfied within the non default rule construct. This can lead to the duplication of vias defined in the LEF if the user wants to use the same non default via for both signal and power net routing. This issue is addressed in LEF 5.6 by the USE VIA syntax.

Virtuoso Studio design environment ASCII Technology File Definition:

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

constraintGroups(
   ( "LEFDefaultRouteSpec " nil
      interconnect(
      ( "validLayers" tx_layerNames ... )
      ( "validVias" tx_viaNames ... )
      ) ;interconnect

      spacings(
      ( minWidth tx_layer1 g_width )
      ( minSpacing tx_layer1 g_spacing )
      ( minWireExtension tx_layer1 g_extension )
      ) ;spacings
   ) ;LEFDefaultRouteSpec
) ;constraintGroups


layerDefinitions(
   techLayers(
   ( tx_layerName tx_layer# tx_abbreviation )
   )
   techLayerProperties(
      (edgeCapacitance tx_layer g [tx_layer] _value )
      (resistancePerCut tx_layer [tx_layer] g_value )
      (areaCapacitance tx_layer [tx_layer] g_value )
   ) ;techLayers
) ;layerDefinitions
```

The constraint group name is derived from the non default rule.

Vias that are defined as stackable in `prNonDefaultRule` map to two rules: `stackable` for the two via layers and the minSameNetSpacing spacing constraint for the two via layers.

DFII on CDB ASCII Technology File Definition:

```
prRules(
   prNonDefaultRules(
   ( t_ruleName
      ( ( lt_layer g_width g_spacing [g_notch] [g_wExt] [g_cap] [g_res]
       [g_edgeCap] ) ... )
    ( t_viaName ... )
   [ ( ( lt_layer1 lt_layer2 n_minSpace g_stack )... ) ]
   [ ( properties )...] )
   ...
   prViaTypes(
      ( l_cellView t_viaType )
   ...
   ) ;prNonDefaultRules
) ;prRules
```

Not a default via

OpenAccess Constraint:

Class: `oaSimpleConstraint`
Constraint type: `oacValidRoutingVias`
`oaConstraintGroup` (named after rule)

LAYER: `oacValidRoutingLayers`
WIDTH: `oacMinWidth`
SPACING: `oacMinSpacing`
WIREEXTENSION: `oacMinWireExtension`
`RESISTANCE:` cLefLayerRes if not existing
`CAPACITANCE:` cLefLayerCap if not existing
`EDGE CAPACITANCE:` cLefLayerEdgeCap is not existing
`VIA: oacValidRoutingVias`
`viaDefs: oaCustomViaDef ()`
oaCustomViaDef

Conversion of Technology File leRules Class

CDB	Virtuoso Studio design environment
leRules leLswLayers	leRules leLswLayers

Changes:

No change.

Conversion of Technology File lxRules Class

CDB	Virtuoso Studio design environment
lxRules lxExtractLayers lxNoOverlapLayers lxMPPTemplates	

Note: This class has been deprecated and will be removed in an upcoming release. Cadence recommends that you remove it from your technology files.

Changes:

This section describes the conversion of the `lxRules` section of the CDB technology file to an the Virtuoso Studio design environment constraint groups. The `lxExtractLayers` and `lxNoOverlapLayers` sections will be converted to constraints belonging to a constraint group named "`virtuosoDefaultExtractorSetup`". The `lxMPPTemplates` section is mapped to the `multipartPathTemplates` section in the `devices` section.

Layer Purpose Based lxRules on page 325

Extract Layers on page 326

No Overlap Layers on page 327

Multipart Paths on page 328

Layer Purpose Based lxRules

For information on layer-purpose based lxRules, see Specifying Layer-Purpose Pairs in the validLayers Constraint in the *Virtuoso Layout Suite XL User Guide*.

Extract Layers

The lxExtractLayers rule that earlier provided the list of extractable layers has now been replaced by the validLayers constraint. For more information, see Connectivity Rules in the *Virtuoso Layout Suite XL User Guide*.

No Overlap Layers

The `lxNoOverlapLayers` rule consists of a list of layer pairs which define which pairs of layers should not overlap.

Note: The `lxNoOverlapLayers` rule is no longer supported.

For information about the connectivity rules currently supported, see Connectivity Rules in the *Virtuoso Layout Suite XL User Guide*.

Multipart Paths

The `lxMPPTemplates()` section describes the structure of multipart paths which provide an way of creating guard rings or developing Pcells.

Virtuoso Studio design environment ASCII Technology File Definition:

```
tcCreateCDSDeviceClass( )
multipartPathTemplates(
;( name [masterPath] [offsetSubpaths] [encSubPaths] [subRects] )
;
; masterPath:
; (layer [width] [choppable] [endType] [beginExt] [endExt] [justify]
[offset]
; [connectivity])
;
; offsetSubpaths:
; (layer [width] [choppable] [separation] [justification] [begOffset]
[endOffset]
; [connectivity])
;
; encSubPaths:
; (layer [enclosure] [choppable] [separation] [begOffset] [endOffset]
; [connectivity])
;
; subRects:
; (layer [width] [length] [choppable] [separation] [justification]
[space] [begOffset] [endOffset] [gap]
; [connectivity] [beginSegOffset] [endSegOffset])
;
; connectivity:
; ([I/O type] [pin] [accDir] [dispPinName] [height] [ layer]
; [layer] [justification] [font] [textOptions] [orientation]
; [refHandle] [offset])
;
;( ------------------------------------------------------------------ )
...
) ;multipartPathTemplates
```

DFII on CDB ASCII Technology File Definition:

```
lxMPPTemplates(
( t_mppTemplateName
l_template
) ; end of template
   ...
) ;end of all lxMPPTemplates Rules
; l_template arguments
(
   l_masterPathArgs
   [l_offsetSubpathArgs...]
   [l_enclosureSubpathArgs...]
   [l_subrectangleArgs...]
) ; end of template argument lists
```

OpenAccess Constraint:

N/A

Conversion of Technology File devices Class

CDB	Virtuoso Studio design environment
devices tcCreateCDSDeviceClass symContactDevice ruleContactDevice symEnhancementDevice symDepletionDevice symPinDevice symRectPinDevice cdsViaDevice cdsMosDevice tcCreateDeviceClass tcDeclareDevice	devices tcCreateCDSDeviceClass cdsMosDevice ruleContactDevice multipartPathTemplates

CDB	Virtuoso Studio design environment

Note: These constructs have been deprecated and will be removed in an upcoming release: symContactDevice, symDepletionDevice, symEnhancementDevice, and symPinDevice. symRectPinDevice. Cadence recommends that you remove it from your technology files.

Changes:

Vias as First Class Objects - Updating C and SKILL Code on page 331

Differences Between CDB and the Virtuoso Studio design environment Vias on page 332

Standard Via on page 344

Standard Via Definition on page 344

Custom Via Definition on page 346

Mapping of CDB Devices on page 334

cdsViaDevice to standardViaDefs Mapping on page 335

Mapping CDB Symbolic Device Masters on page 340

Via Specifications on page 348

multipartPathTemplates: See Multipart Paths on page 328

Also see Technology File Via Definitions and Via Specifications on page 343

Create Contact Command Obsolete

The *Create – Contact* command has been changed to the *Create – Via* command. The *Create – Via* command checks the `validRoutingVias` in a constraint group, if defined, which will in turn reference a subset of technology file defined `viaDefs`. If the `validVias` constraint is not defined, all `viaDefs` are used.

Vias as First Class Objects - Updating C and SKILL Code

In the Virtuoso Studio design environment, vias are first class objects. Do not use the *Create – Instance* command to place vias. Existing user code that refers to, or searches for vias as instances must be changed to search for vias as first class objects. For example, your

user code might iterate through instances and use heuristic code to determine which are vias. In the Virtuoso environment, you can directly query cellviews for vias.

Differences Between CDB and the Virtuoso Studio design environment Vias

- CDB
  - Contacts are predefined Pcells (syContact, syEnhContact, cdsViaDevice), or user defined Pcells, or fixed cellviews.
  - Can be placed as instances.
  - Recognized as devices of certain classes in technology file.
  - For Preview and LEF/DEF, via masters defined in the prRules sections of the technology file.
  - Example of cdsViaDevice on CDB:

```
cdsViaDevice(
; (name cutLayer cutPurpose layer1 purpose1 layer2 purpose2 row column
; origin stackedVias cutLayerW cutLayerL xCutSpacing yCutSpacing
; layer1XDirOverride layer1YDirOverride layer2XDirOverride
; layer2YDirOverride layer1Dir layer2Dir
; layer1XDefOverride layer1YDefOverride layer2XDefOverride
; layer2YDefOverride l_implantLayers diffSpacing abutClass)
(m1_m2_new via1 drawing metal1 drawing metal2 drawing 1 1
centerCenter t _NA_ _NA_ _NA_ _NA_
0.5 0.5 _NA_
_NA_ "rt" ""
0.25 _NA_ _NA_
_NA_ nil nil _NA_)
```

- Virtuoso Studio design environment
  - Via definitions associated with an OpenAccess `oaVia` object.
  - If an instance of a master referenced by a customViaDefs is placed, the instance will not be seen a via by applications.
  - Defined in the `viaDef` section of the technology file and referenced by the `validVias` constraint defined in a constraint group.
  - Example of cdsViaDevice on OpenAccess:

```
cdsViaDevice(
; (name cutLayer cutPurpose layer1 purpose1 layer2 purpose2 row column
; origin stackedVias cutLayerW cutLayerL xCutSpacing yCutSpacing
; layer1XDirOverride layer1YDirOverride layer2XDirOverride
; layer2YDirOverride layer1Dir layer2Dir
; layer1XDefOverride layer1YDefOverride layer2XDefOverride
; layer2YDefOverride l_implantLayers diffSpacing abutClass)
```

```
(m1_m2_new via1 drawing metal1 drawing metal2 drawing 1 1
centerCenter t _def_ _def_ _def_ _def_
0.5 0.5 _def_
_def_ "rt" ""
0.25 _def_ _def_
_def_ nil nil _def_ "")
(via2 cont drawing poly drawing metal1 drawing
1 1 centerCenter t _def_ _def_ _def_ _def_
0 0 0 0
"" ""
0.3 0.3 _def_ _def_
(
(("well" "drawing") 1)
(("nimplant" "drawing") 1.5)
)
nil
0 "")
)
```

Note: Some parameters have different meanings in OpenAccess and CDB. Using CDB parameters while migrating a technology file from CDB to OpenAccess can cause unexpected results after migration.

On CDB	On OpenAccess
_NA_: default	_def_: use the default value
	_NA_: not needed or not appropriate
	nil: boolean false

Mapping of CDB Devices

If your CDB design contains vias with master cellviews defined in the design library rather than the technology library, you must list the vias in a mapping file prior to migration

Important

If any device is defined in the prRules section, the device is mapped to `customViaDefs` and if possible, to a `standardViaDefs`, based on the device definition. If `customViaDefs` are created, then a fixed cellview will also be generated. If the original is a Pcell, the fixed cell is based on the default parameters.

- `cdsViaDevice` device maps to `standardViaDefs` if possible (see below), otherwise maps to a `customViaDefs`.

  If all parameters can be mapped, a `cdsViaDevice` device is mapped to a `standardViaDefs`. If the `abutmentClass`, `diffusionSpacing`, and `implantLayers` have values the `cdsViaDevice` device is mapped to `customViaDefs`.

- *`userDeviceContact`* device maps to `customViaDefs` only if `layer1` and `layer2` are defined and the function type is `contact` or `substrateContact`.

- The `syEnhContact` is a user defined class and maps to `customViaDefs` only if `layer1` and `layer2` are defined and the function type is `contact` or `substrateContact`.

- `ruleContact` device maps to a `customViaDefs`. The `ruleContact` is a system device class with a function type of `contact`.

- `syContact` device maps to `standardViaDefs`.

  See Mapping CDB Symbolic Device Masters on page 340 for information about retaining cellviews during translation.

- `syPin` and `syRectPin` devices and definitions are deleted from the technology file. The cellviews are by default deleted and any instances as pins of the cellview are converted to rectangles and pins.

- `syEnhancement` and `syDepletion` classes and the associated device definitions are deleted from the technology file.

cdsViaDevice to standardViaDefs Mapping

cdsViaDevice	standardViaDefs
name	viaDefName
cutLayer	cutLayer
cutPurpose	drawing (Not user definable. Mapped to drawing.)
layer1	layer1
purpose1	drawing (Not user definable. Mapped to drawing.)
layer2	layer2
purpose2	drawing (Not user definable. Mapped to drawing.)
row	cutRows
column	cutColumns
origin	origOffset
stackedVias	N/A (Use the stack via constraint stackable)
cutLayerW	cutWidth
cutLayerL	cutHeight
xCutSpacing	cutSpacing Component of the cutSpacing origin offset. Specified as a vector, with X spacing value in OpenAccess.
yCutSpacing	cutSpacing Component of the cutSpacing origin offset. Specified as a vector, with Yspacing value in OpenAccess.

cdsViaDevice	standardViaDefs

`layer1XDirOverride`
`layer1YDirOverride`
`layer2XDirOverride`
`layer2YDirOverride`
`layer1XDefOverride`
`layer1YDefOverride`
`layer2XDefOverride`
`layer2YDefOverride`
`layer1Dir`
`layer2Dir`
	Not directly mapped. Parameters are used to determine the `layer{1,2}Offset` and `layer{1,2}Enc`.

`implantLayer1`
`implantLayer2`
	`implantLayer1 (implant1Enc)`
	`implantLayer2 (implant2Enc)`
	where `implantLayer = list((implantLayer n_encWidth))`
	When a value is specified, the device is mapped to a customViaDefs.

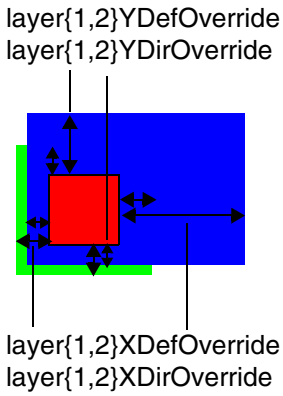
`diffSpacing`	Parameter not mapped. When a value is specified, the device is mapped to a customViaDefs.
`abutClass`	Parameter not mapped. When a value is specified, the device is mapped to a customViaDefs.

Loading ASCII cdsViaDevice

Standard Via Layer Enclosures

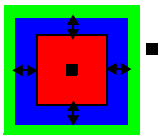
Defining the layer enclosure for standard vias is based on the origin of the via, rather than the enclosure of metal overlap distance.

CDB CDSViaDevice



Virtuoso Studio design environment Standard Via

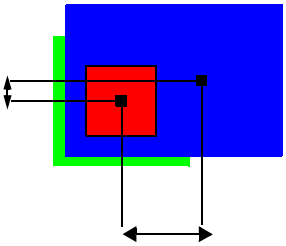
Determines the size of layers



layer{1,2}Enc
Specifies the layer{1,2}
enclosure as an X/Y vector

Determines the location or offset of layers

layer{1,2}OffsetY
References a vector object with
X spacing value



layer{1,2}OffsetY
References a vector object with
X spacing value

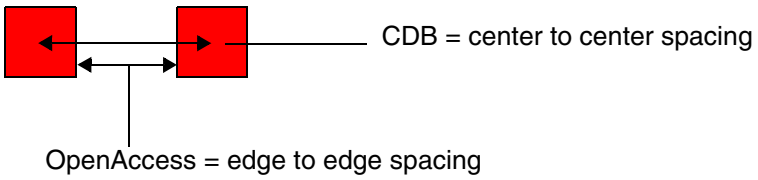
The CDB x and y layer{1,2}{Dir,Def}Overrides mapped to the layer{1,2}Enc oaVector object whose x and y values indicate the x and y offsets respectively.

The origin (justification) maps to the layer{1,2}Offset{X,Y}. In cases where layer1 and layer2 are not aligned on the center of the cut, the layer offset parameters specify the amount of offset between the center of the layer and the center of the cut. In example above, the offset is added to the coordinates of the corresponding rectangle, so a positive offset moves the rectangle up and to the right.

Note:

syContact to standardViaDefs Mapping

syContact	standardViaDefs
`name`	`viaDefName`
`viaLayer`	`cutLayer`
`viaPurpose`	`drawing` (Not user definable. Mapped to `drawing`.)
`layer1`	`layer1`
`purpose1`	`drawing` (Not user definable. Mapped to `drawing`.)
`layer2`	`layer2`
`purpose2`	`drawing` (Not user definable. Mapped to `drawing`.)
`w`	`cutWidth`
`l`	`cutHeight`
`row`	`cutRows`
`col`	`cutColumns`

`xPitch, yPitch`
Note: Center to center spacing
	`cutSpacing`
	`cutSpacing x = xPitch - cutWidth`
	`cutSpacing y = yPitch - cutHeight`
	Specifies the spacing as a vector, with X or Y spacing value.
	Note: Edge to edge spacing.
	 

`xBias, yBias`	`origOffset`
`xBias, yBias` `encByLayer1`	`layer1Enc x = encByLayer1` `layer1Enc y = encByLayer1`

syContact	standardViaDefs
`xBias, yBias` `encByLayer2`	`layer2Enc.x = encByLayer2` `layer2Enc.y = encByLayer2`

`implantLayer1`
`implantLayer2`
	`implant1 (implant1Enc)`
	`implant2 (implant2Enc)`
	where `implantLayer = list(ltx_implantLayer n_encWidth)`

`resistancePerCut` store as property in `syContact`	if `resistancePerCut` exist as a device property the value is mapped to `resistancePerCut` of the via definition. Also see, Translation of the resistancePerCut Property on page 108.
`legalRegion`	store as property in `standardViaDefs` where `legalRegion = list(t_inOut ltx_wllLP)` **Note:** `legalRegion` is a legacy rule which is no longer supported by applications.

Mapping CDB Symbolic Device Masters

Symbolic device masters (`syContact`, `syPin`, and `syRectPin`) are obsolete in the Virtuoso environment.

- `syContact` device definitions are deleted from the technology file, device masters are by default deleted from disk (Keeping Device Masters), and the instances are mapped to vias of type `standardViaDefs`.
- `syPin`, and `syRectPin` device definitions are deleted from the technology file, device masters are by default deleted from disk (Keeping Device Masters), and the instances are converted to rectangular shapes with pins.

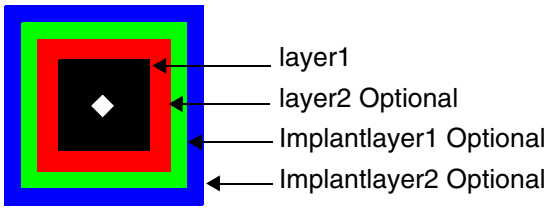
How CDB Symbolic Devices syPin and syRectPin are Converted

A CDB `syPin` or `syRectPin` can be made up as a one layer pin or as a two layer pin and can optionally have implant layers. These pins are converted to shapes as rectangles with pins.

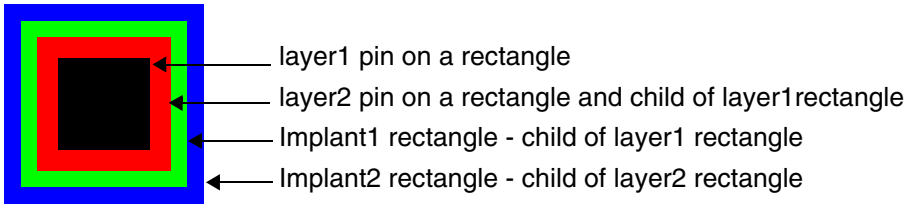
When a two layer pin is converted, the first layer is created as a rectangle shape with a pin, the second layer is created as a rectangle shape that is a child of the first shape with a

separate pin. Any optional implant layers are created as children of their respective parents, first or second layer rectangles. All pins are strongly connected.

CDB syPin or syRectPin



Virtuoso Studio design environment shapes as rectangles on layers



Keeping Device Masters

To preserve existing customer data integrity during migration from CDB to OpenAccess, a limited preservation of symbolic device masters (`syContact`, `syPin`, and `syRectPin`) is possible.However, this does not change any existing mapping or current technology file processing. The devices will be deleted from the technology file, only the cellviews will be retained. The device masters can not be updated or recreated and a query to the technology file does not return any symbolic device information. The `keepdevicemasters` option is recommended as a temporary solution if you have SKILL Pcells that instantiate `syContacts`, `syPins`, `syRectPins`, and in some cases `cdsViaDevices`.

Updating SKILL Code

SKILL code that references these devices must be updated and instances of the `syContact`, `syPin`, and `syRectPin` devices deleted. SKILL code updates include;

- Update any applicable calls of `dbCreateParamInst()` and `dbCreateParamInstByMasterName()` to `dbCreateVia()`.

  Change involves `syContact` devices being mapped to `standardViaDefs` and masters deleted. For more information, see syContact to standardViaDefs Mapping on page 338.

- Use `dbCreateRect()` and `dbCreatePin()` functions to create pins.

Change involves `syPin`, and `syRectPin` device masters are deleted and the instance converted to rectangle with pin figures.

- For `cdsViaDevices`, use the `dbCreateVia()` function, and reference either `standardViaDefs` or `customViaDefs`.

  Change involves `cdsViaDevice`: maps to `standardViaDefs` if possible (see below), otherwise maps to a `customViaDefs`. For more information, see Mapping of CDB Devices on page 334.

- The function `leCreateAutoInstPin` was used to create `syPin` devices, which are obsolete. In the Virtuoso environment, pins must be created as rectangle shapes using `leCreateAutoPins`. The following warning message is issued when attempting to create `syPin` devices using `leCreateAutoInstPin`.

```
*WARNING* syPin devices are no longer supported in OA 2.2, use leCreateAutoPin().
```

Note: The function `leCreateAutoInstPin` was used only to create instances of `syPin` devices, which are obsolete. This change does not imply that instances can not be pins. Pins can be created as instances and you can specify an instance as a pin figure.

Obsolete Environment Variables

The environment variables `pinType` and `pinLength` are obsolete in Virtuoso applications.

Technology File Via Definitions and Via Specifications

Most CDB technology file contacts device definitions are translated to via definitions. Technology file via master definitions (`viaDefs`) consist of two types of vias; `customViaDefs` and `standardViaDefs`.

See Conversion of Technology File devices Class on page 330.

Standard Via on page 344

Standard Via Variant on page 345

Custom Via on page 346

Custom Via Variant on page 347

Via Specifications on page 348

Standard Via

Note: In the Virtuoso environment, the masters of standard vias are not associated with a library, cell, or view name. A NULL value is returned when attempting to use `dd` function calls to return the master ID of standard vias.

The stand`ardViaDefs` has a unique name and is associated with two metal layers, a cut layer, and a list of via parameters with default values. It is an object that is predefined by the database. Via definitions (`viaDefs`) correspond to the OpenAccess `oaViaDef` class. Standard vias are by default created using the layer purpose *drawing*. In order for the vias to be displayed, the layer purpose for via layers must be *drawing* and must exist in the technology file. You must have *drawing* lpp for each layer defining viaDefs.

Standard Via Definition

Virtuoso Studio design environment ASCII Technology File Definition:

```
viaDefs(
standardViaDefs(
    ( t_viaDefName tx_layer1 tx_layer2
    ( t_cutLayer n_cutWidth n_cutHeight [n_resistancePerCut] )
    ( x_cutRows x_cutCol ( l_cutSpace ) )
    ( l_layer1Enc ) ( l_layer2Enc ) ( l_layer1Offset )
   ( l_layer2Offset ) ( l_origOffset )
    [tx_implant1 (l_implant1Enc) [tx_implant2 (l_implant2Enc]]
    )
) ;standardViaDefs
) ;viaDefs
```

DFII on CDB ASCII Technology File Definition:

See Mapping of CDB Devices.

OpenAccess Constraint:

Class: `oaViaDefs`
Constraint type: `oaStdViaDef`

LEF Syntax:

See prViaTypes

Standard Via Variant

A standard via variant is like a standard via but with predefined parameter values.

Virtuoso Studio design environment ASCII Technology File Definition:

```
viaDefs(
  standardViaVariants(
; (viaVariantName viaDefName (cutLayer cutWidth cutHeight)
; (cutRows cutCol (cutSpace))
; (layer1Enc) (layer2Enc) (layer1Offset) (layer2Offset) (origOffset)
; (implant1Enc) (implant2Enc)
; (----------------------------------------------------------------)
  ( t_name t_viaDefName ( t_cutLayer f_cutWidth f_cutHeight )
      ( x_cutRows x_cutCols ( l_cutSpace ) )
      ( l_layer1Enc ) ( l_layer2Enc ) ( l_layer1Offset )
      ( l_layer2Offset ) ( l_origOffset )
      ( l_implant1Enc ) (l_implant2Enc )
   )
 ) ;standardViaVariants
) ;viaDefs
```

OpenAccess Constraint:

Constraint type: `oaStdViaVariant`

Custom Via

The `customViaDefs` has an unique name and is associated with a master cellview and two layers.

A customViaDefs is a special purpose via. The `lib/cell/view` is declared in the technology file for the custom via as in the `M3_M2` example below.

```
customViaDefs(
;( viaDefName libName cellName viewName layer1 layer2 resistancePerCut)
;( ---------- ------- -------- -------- ------ ------ ----------------)
( M3_M2_layout  tutorial  M3_M2  layout  METAL2   METAL3  0.0)
( M2_M1  sample           M2_M1  via     METAL1   METAL2  0.5)
)
```

Custom Via Definition

Virtuoso Studio design environment ASCII Technology File Definition:

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

DFII on CDB ASCII Technology File Definition:

See Mapping of CDB Devices.

OpenAccess Constraint:

Class: `oaViaDefs`
Constraint type: `oaCustomViaDefs`

LEF Syntax:

See prViaTypes

Custom Via Variant

A custom via variant is like a custom via but with predefined parameter values.

Virtuoso Studio design environment ASCII Technology File Definition:

```
viaDefs(
  customViaVariants(
; (viaVariantName viaDefName (paramName paramValue) ...)
; (-------------------------------------------------)
  ( t_name t_viaDefName ( t_paramName t_paramValue ... ) )
 ) ;customViaVariants
) ;viaDefs
```

OpenAccess Constraint:

Constraint type: `oaCustomViaVariant`

Via Specifications

Via specifications (`viaSpecs`) store an array of via definitions. The via specifications objects are defined by the specified name, physical layers and via definitions array. Via specifications (`viaSpecs)` correspond to the OpenAccess `oaViaSpec` class.

A viaSpec can be either an `oaViaDefArrayValue` or an `oaViaDef2DTblValue`. An `oaViaDefArrayValue` is a list of viaDefs, and the `oaViaDef2DTblValue` contains the width and viaDefArray information. Only one viaSpec is allowed per layer pair.

The rows of the table will correspond to the layer1 minimum and maximum width pairs, the columns will correspond to the same information for layer2. viaSpecs are not part of a constraint group.

The following is the technology file syntax for the Virtuoso Studio design environment via specifications.

- Virtuoso Studio design environment viaSpecs

```
viaSpecs(
   (tx_layer1   tx_layer2 ("lt_viaDefNames")
      (
      (g_layer1MinWidth   g_layer1MaxWidth
      g_layer2MinWidth    g_layer2MaxWidth ("lt_viaDefNames")
      )
   )
)
```

Technology File Site Definitions

Virtuoso Studio design environment ASCII Technology File Definition:

```
siteDefs(
scalarSiteDefs(
;( name type width height [symX][symY][symR90])
 (----- ---- ----- ------ ----- ------ ------)
 ( t_name t_type n_width n_height [g_symmInX] [g_symInY] [g_symmInR90] )
) ;scalarSiteDefs
arraySiteDefs(
; ( name type
; ( (siteDefName dx dy orientation) ... )
; [symX] [symY] [symR90] )
  ( t_name t_type
    ( (t_siteDefName f_dx f_dy s_orient)... )
    [g_symmInX] [g_symInY] [g_symInR90]
  )
) ;arraySiteDefs
); siteDefs
```

DFII on CDB ASCII Technology File Definition:

When a cellview contains the property `prCellType` with the value `site`, the cellview will be mapped to a `siteDef`.

OpenAccess Constraint:

Class: `oaSiteDef`
Class: `oaSiteDefArray`, `oaScalarSiteDef`

In the Virtuoso Studio design environment, two types of site definitions (`siteDefs`) exist, scalarSiteDefs and arraySiteDefs. In LEF 5.6, macros can now have multiple site definitions for gate-array standard cells. The oaSiteDefArray class has been introduced in OpenAccess 2.2 to support these multiple site definitions.

- LEF 5.6 Syntax

```
SITE siteName
   CLASS {PAD | CORE} ;
   [SYMMETRY {X | Y | R90} ... ;]
   [ROWPATTERN {existingSiteName siteOrient} ... ;]
   SIZE width BY height ;
END siteName
```

The arraySiteDefs syntax stores the LEF 5.6 syntax defining the pattern of sites. In LEF, the name of the site and the orientation is required. However in OpenAccess 2.2 specification of both an orientation and location is allowed, which provides for spacings between the sites in a row to be defined.

The width and height values from a LEF file is not stored in the technology file. The width and height of an array of sites is calculated from the pattern definition. When a LEF file is read into the Virtuoso environment, if the width and height are specified it is ignored. When a LEF file is read out of the Virtuoso environment, the correct width and height is calculated from the pattern definition.

For each arraySiteDef, the pattern of sites along the row can be specified as a series of entries in the technology file. The location of the sites is defined by the dx and dy values. The dx and dy values specify the offset in the x and y directions respectively from the origin (origin is defined as the lower left corner of the array) of the site array. There is no special order for these entries. Each entry defines a particular site which must be defined as a scalarSiteDef with a corresponding list of transformations. The allowed orientations must be one of R0, R90, R180, R270, MX, MY, MXR90, and MYR90. The symmetry arguments are one of, symX, symY, and symR90. The default symmetry is `nil`.

5

Changes to Environment Objects and Shapes

Differences Between CDB and Virtuoso Studio Design EVirtuoso Studio design environmentnvironment Objects

In DFII on CDB applications, layer-purpose pairs (LPPs) are used to represent the purpose of a shape. For example, a rectangle with the conventions of the LPP, `metal1 boundary` is used to define the shape for the specific purpose of a boundary. When a shape using conventions originates from CDB and is translated into a Virtuoso Studio design environment application, these conventions of shapes on specific LPPs do not apply.

When translating data from CDB to the Virtuoso Studio design environment, shapes such as boundaries and blockages are converted to objects. The selection, display, and editing is object based. These objects have attributes and can be created and edited through a GUI or through SKILL. Although these objects do not use LPPs to represent their purpose, they are assigned to an LPP to allow display.

Objects in place and route tools such as SoC Encounter, such as a rows and track patterns are supported in the Virtuoso environment. They are assigned LPPs for display, and depending on the application, creating and editing is allowed.

Below is a list of objects supported in the Virtuoso Studio design environment.

Interconnect Objects

Rows

Sites

Blockages

Boundaries

Clusters

Track Patterns

For a full listing of each class in the OpenAccess API, go to the URL http://openeda.si2.org.

For a description of CDB conventions for these objects, see CDB and Preview Conventions for Identifying Objects.

Virtuoso Studio design environment Objects

Throughout the documentation concerning there are references to objects according to how they are used in an application or flow. Below are the list of terms used through out the Cadence Help documentation and their descriptions.

Term Used	Definition
DFII CDB shape based data	Shapes which use conventions, for example a rectangle on *metal1 boundary* to define its purpose as a boundary.
Standard OpenAccess objects, Native OpenAccess data	Objects such as a row, boundary, blockage, etc., that do not use conventions to define their purpose.
Place and route objects	Objects that originate from a place and route tool which do not use conventions to define their purpose.

Interconnect Objects

Path

A single layer object. Paths can be created using the *Create - Path* command or can be optionally selected as an element to be used when creating wires. Paths support *anyAngle* mode. When used to do non-orthogonal routing, paths can lead to off-grid vertices being created, depending on the width and angle being used.

Wire

A contiguous string of paths, segments (pathSegs), and vias used to connect same net instance terminals and pins of the cellview. Wires are not directly associated with a database object.

For information about creating wires, see Creating Wires in the *Virtuoso Interactive and Assisted Routing User Guide*.

pathSegs

A specialized shape-based object represented by a specified width, a layer-purpose pair, and a two-point routing segment with a style associated with each of the points. Pathsegs can be

orthogonal or diagonal and can be used to create on-grid 45 degree wires. Custom end styles containing octagonal edges are used in diagonal pathSegs to pull all vertices on grid.

For information about accessing pathSeg attributes, see pathSeg Attributes in the *Virtuoso Design Environment SKILL Reference*.

Segment of Path

Section of an individual path object.

Route

A route is associated with the OpenAccess object oaRoute. Routes are containers that can hold the objects pathSeg, vias, and guides.

Guide

A guide specifies a connection between two objects that are intended to be wired together. Routing nets usually replaces guides with wires, or routes. Currently, guides are only supported at the API level.

Via

In CDB, a contact is implemented as a pre-defined Pcell, which is a special instance. Virtuoso Studio design environment applications will not recognize a via placed as a instance. In the Virtuoso environment, vias are first class objects. Vias and instances (specifically contacts placed as instances) are considered two different types of objects in OpenAccess. Vias correspond to the OpenAccess `oaVia` class and instances correspond to the `oaInst` class. The CDB menu item *Create – Contact* is replaced with *Create – Via* in the Virtuoso environment.

Existing user code that refers to, or searches for vias as instances must be changed to search for vias as first class objects.

All CDB contacts defined in technology file are translated into via definitions. Via definitions (`viaDefs`) define via cellviews in a technology database. There are two types of via definitions; `customViaDefs` and `standardViaDefs`.

For information about how vias are mapped, see Mapping of CDB Devices on page 334.

If your CDB design contains vias with master cellviews defined in the design library rather than the technology library, you must list the vias in a mapping file and refer to the mapping file when migrating the library from CDB to OpenAccess.

Standard Via

The standard via references a standard via definition which is implemented by the oaStdViaDef object. The standardViaDefs has an unique name and is associated with two layers and a list of via parameters with default values. It is an object that is predefined by the database.

Custom Via

A custom via is an instance of a custom via definition which is implemented by the `oaCustomViaDef` object. A custom via has an unique name and is associated with a via master. The via master may be parameterized and its parameters are specified through the use of an oaParamArray object.

For information, see Via Attributes in the *Virtuoso Design Environment SKILL Reference* and Via Object C Functions in the *Cadence Integrator's Toolkit Database Reference*.

Rows

A row represents a location for the placement of standard cells, macros, or device and is a repetition of the site element in a given direction (horizontal or vertical). Each row is associated with a siteDef. All sites in one row must be the same size. The display of row objects can be turned on or off and the snapping of instances to row can be controlled. Rows are drawn using the display attributes for LPP *row boundary*.

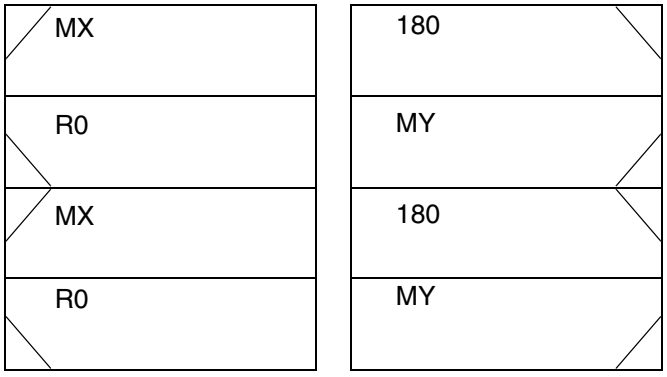
Rows define a pattern of a sites and are generated by the ROW statement in a DEF file. Rows are drawn as a rectangle with a diagonal line in the same corner as the origin which shows the row orientation.

Two types of rows exist; horizontal and vertical rows. Horizontal rows are of the orientation `R0`. Vertical rows are of the orientation R90.

There are four possible site orientations, `R0`, `MX`, `180`, and `MY` for horizontal and vertical rows. It is the orientation of the row with the site which defines the type of row.

Table 5-1 Horizontal Rows

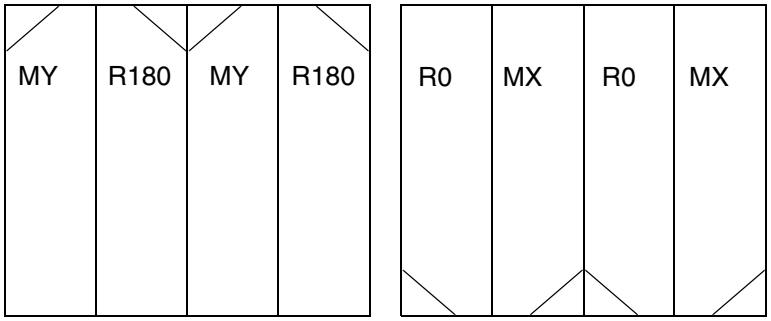
Row Orientation	Site Orientation
`R0`	`R0`
`R0`	`MX`
`R0`	`R180`
`R0`	`MY`



Abutted Horizontal Rows
Orientation R0

Table 5-2 Vertical Rows

Row Orientation	Site Orientation
R90	R180
R90	R0
R90	MY
R90	MX



Abutted Vertical Rows
Orientation R90

Creating Rows Using SKILL

Rows can be created using DEF In and though SKILL. The following example creates a horizontal row, with the site orientation of `MX`, at the origin point of `0:0`, with 50 sites using SKILL.

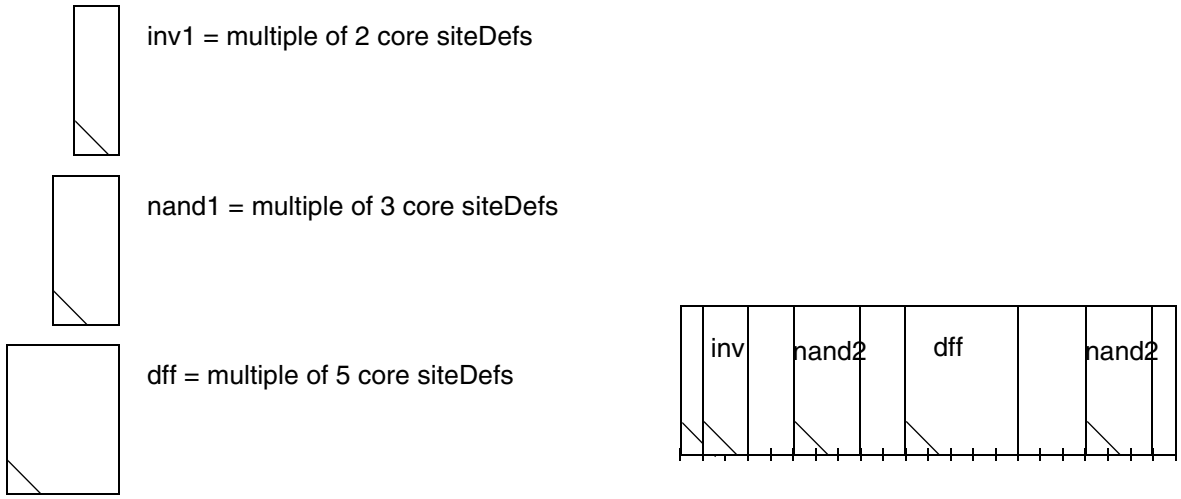
```
cv = geGetEditCellView()
tf = techGetTechFile(cv)
site1 = techCreateScalarSiteDef(tf "site1" "core" 5 5)
dbCreateRow(cv site1 "row1" list(0 0) 50 "MX" "R0")
```

For more information, see Row Attributes and Row Functions in the *Virtuoso Design Environment SKILL Reference* and Row Headers and Rows in the *Cadence Integrator's Toolkit Database Reference*.

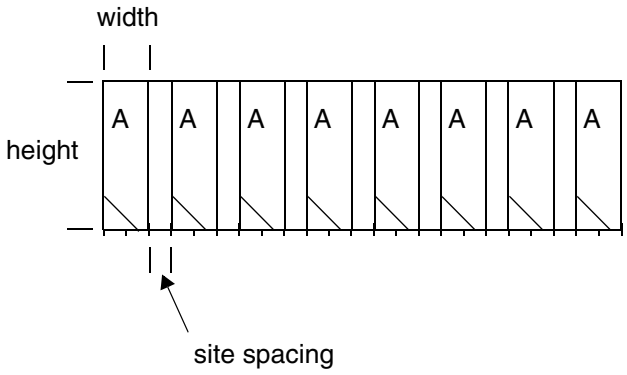
Sites

Site definitions (`siteDefs`) are technology objects which are the basic elements from which placement rows are constructed.

Core sites are used for standard cell placement in rows. Cell widths must be a multiple of the site width. If there are multiple standard cell heights, a site must be defined for each standard cell height. A row is basically a one dimensional array of a given site. A scalar site definition defines the width, height, and allowable symmetry of each placement location in a row.



The array site definitions, are a two dimensional array of siteDefs. T

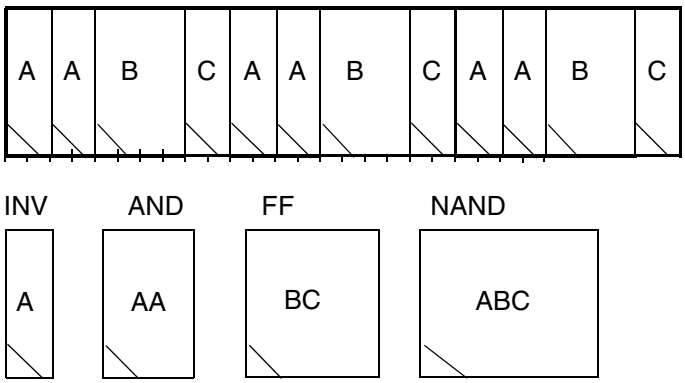


The siteRef also holds an additional transform parameter that provides the capability of introducing empty 2 dimensional space between sites - which will need to be considered as a sitePattern defines the pre-determined set of sites. This empty space transform is available in both oaScalarSiteDef and in oaArraySiteDef.

The array site definition defines a pattern of scalar sites. It has a pre-determined set of sites that are used to place a set of cells as part of a row. This type of site is primarily used in gate-array designs. The array site definition references the sitePattern definition to determine to the set of sites. Each sitePattern is a named site reference (siteRef) to an oaScalarSiteDef.

The pattern that is specified will determine whether a cell can be placed or not. In the following example the pattern is AABC. You can place cells with one of the following patterns.

"A" "B" "C" "AA" "AB" "BC" "AAB" "ABC" "AABC"



Note: When a CDB cellview contains the property `prCellType` with the value of `site`, the cellview will be mapped to a `siteDef` on the OpenAccess database. The technology file `siteDefs` section will be updated accordingly.

For more information, see SiteDef Attributes in the *Virtuoso Design Environment SKILL Reference* and Site Definition C Functions in the *Cadence Integrator's Toolkit Database Reference*.

Blockages

A blockage indicates an area in a design in which a certain type of object is not allowed to be placed. There are three types of blockages, area blockage, layer blockage, and area halo.

- An area blockage is used to prevent instances from being placed within a specific area.
- A layer blockage represents an area on a given layer where an instance cannot be placed. In Virtuoso, a layer blockage cannot be of type placement.
- An area halo is associated with an instance or a prBoundary to represent an extended area around the master's prBoundary. An area halo extends the area around a instance or a prBoundary in four orthogonal directions. The offsets in these directions can be different.

Owners

Different types of blockages can have different owners.

Type of Blockage	Type of Owner	
areaHalo	instance, prBoundary	not optional
areaBlockage	instance, cluster	optional
layerBlockage	instance, cluster	optional

Blockage Types

Area halo blockages can only be of type `placement`. Area and layer blockages support all of the following blockage sub-types except `placement`.

```
routing
placement
wiring
fill
slot
pin
feedthru
Screen
```

For information about accessing blockage attributes, see Blockage Attributes and Blockage SKILL Functions in the *Virtuoso Design Environment SKILL Reference* and Blockage C Functions in the *Cadence Integrator's Toolkit Database Reference*.

Boundaries

The CDB to OpenAccess migration converts CDB boundary shapes on layers to an OpenAccess prBoundary object. All boundary objects are assigned to layer purpose pairs (LPP) to allow display. The LPPs are used for object display and selection and should be set to not valid in the LSW. Meaning the LPPs should not be available for shape drawing from the *Layers* panel of the LSW. Control selection and display of objects through the *Object* panel of the LSW. For a list of required technology file layers for OpenAccess objects, see Required Technology File Layers and Purposes for OpenAccess Objects.

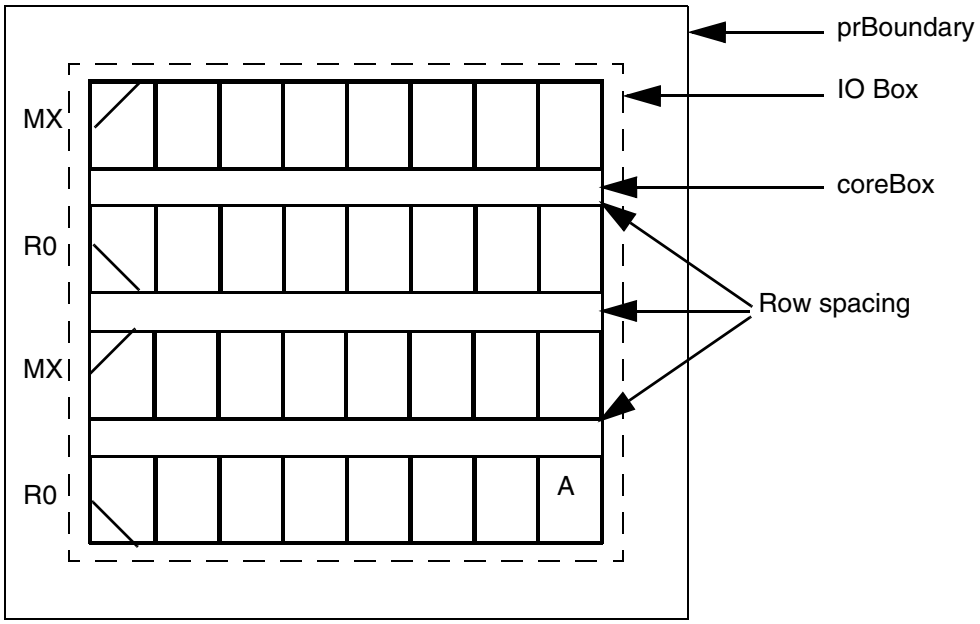
There are four types of boundary objects, area boundary, snap boundary, prBoundary, and cluster boundary.

- An area boundary specifies an area in a block for a specific purpose. Area boundaries are named. A block can have multiple area boundaries, each identified by its name.
- The snap boundary is used by tools to place instances of the cellview in rows. Depending on rotation and symmetry, the appropriate corner is snapped to the site in the row.
- The cluster boundary specifies the area in which a cluster of objects (group of objects) that are associated with a cluster should be placed.
- The prBoundary is used to represent the boundary of a block for place-and-route applications. Two attributes have been added, a coreBoxSpec and an IO box. Both attributes are mainly used by Soc Encounter. Currently, Virtuoso tools do not use the coreBoxSpec and the IO box.

  A coreBoxSpec defines a rectangular grid of sites (a row of rows) within the prBoundary which defines how physical rows, and cells within the rows, can be placed in the prBoundary.

  An IO box, together with the core box, indicates the area for the IO cells.

In the example below, the prBoundary has an IO box and a coreBox. The coreBox has four horizontal rows of siteDef A.



All rows are separated by a distance row spacing. Every odd numbered row is flipped.

The number of rows and row geometry is computed based on the overall bBox of the coreBox and the siteDef of the coreBox. API is available to retrieve the number of rows and bBox of each row.

For information about accessing boundary attributes, see Boundary Attributes and Boundary SKILL Functions in the *Virtuoso Design Environment SKILL Reference* and Boundary C Functions in the *Cadence Integrator's Toolkit Database Reference*.

Mapping Shape Based Boundaries

Shapes such as boundaries on layers are mapped to an OpenAccess prBoundary object. The following describes how to map CDB boundary shapes on specific layers to boundary objects.

In CDB, prBoundaries are shapes which use the convention of LPPs to identify them as boundaries. In CDB you can define spacing rules to specify data to boundary relationships. For example,

```
( minEnclosure( "prBoundary" "boundary" ) ( "nwell" "drawing" ) 0.4 )
( "minEnclosure" "prBoundary" "metal1" 0.3 )
( "minExtension" "prBoundary" "metal3" 0.5)
```

In the Virtuoso Studio design environment, prBoundaries are objects that do not use conventions to define their purpose. When translating data to OpenAccess, prBoundaries defined as shapes on layers, are mapped to prBoundary objects. The constraints Minimum Boundary Extension Constraint and Minimum Boundary Interior Halo Constraint are used to specify the data to prBoundary object relationships. The `minEnclosure` and `minExtension` rules are mapped to `minPRBoundaryInteriorHalo` and `minPRBoundaryExtension`. For example, the rules specified above would map to the Virtuoso Studio design environment as follows.

```
( "minPRBoundaryInteriorHalo" "nwell"  0.4 )
( "minPRBoundaryInteriorHalo" "metal1"  0.3 )
( "minPRBoundaryExtension" "metal3" 0.5 )
```

In CDB there are various LPP conventions used with the rules `minExtension` and `minEnclosure` that define shapes as prBoundaries. Because of this, a mapping file can be used to map the LPPs used in the `minExtension` and `minEnclosure` rules, to the OpenAccess constraints `minPRBoundaryExtension` and `minPRBoundaryInteriorHalo`.

The mapping file is named `.techfilemapping` and should be located in the same directory as the library definition file (`cds.lib`). The file consist of a priority ordered listing of LPPs, which are used to identify the correct shapes to be used when generating the prBoundary object.

Creating a Mapping File

The mapping file should consist of a list of LPPs (purpose is optional) in the order of priority to indicate which layers and possible purpose to use to map shapes on layers to prBoundary objects for each cellview. The first layer specified indicates the highest priority. The last layer specified indicates the lowest priority.

The syntax for boundary priority mapping file is:

```
; A list to search prBoundary for Virtuoso on OA2.2
     techBoundaryLayerPurposePriorities(
     ( t_layerName  [t_purposeName] )
     ...
     )
```

For example:

```
;*********************************************
; A list to search prBoundary for Virtuoso on OA2.2
;*********************************************
techBoundaryLayerPurposePriorities(
;   ( layerName      layerPurpose )
```

```
    ( prBoundary     boundary )
    ( prBoundary     drawing  )
    ( prBoundary              )
    ( metal1         drawing  )
)
```

The two rules `minEnclosure` and `minExtension` which use the LPPs specified in the mapping file are converted to the constraints, `minPRBoundaryExtension` and `minPRBoundaryInteriorHalo` based on the priority of the listed LPPs in the file.

If the of the constraints (`minPRBoundaryInteriorHalo` and `minPRBoundaryExtension`) do not exist in the `foundry` constraint group, a constraint will be created. If a constraint has already been defined but the value is the same, the constraint is used without error or warning. If a constraint has already been defined but with a different value, a warning will be generated. It is assumed that any one cellview will follow only one convention of LPP.

If a mapping file can not be found by the translator, the first LPP used as a default is, `prBoundary boundary`. If the LPP `prBoundary boundary` can not be found, `prBoundary` is used as the default.

Rules Definitions

In DFII on CDB 5.1.41, the rules `minEnclosure` and `minExtension` must be defined as ordered spacing rules and requires the layer name be either a layer name pair or a layer-purpose pair.

For example, the following rules are defined correctly for mapping purposes.

```
( minEnclosure ( "prBoundary" "boundary" ) ( "metal1" "drawing" ) 0.3 )
( "minEnclosure" "prBoundary" "metal2" 0.3 )
( "minEnclosure" "prBoundary" ( "metal3" "all" ) 0.4 )
```

The rules specified above would map to the Virtuoso environment as follows:

```
( "minPRBoundaryInteriorHalo" "metal1"  0.3 )
( "minPRBoundaryInteriorHalo" "metal2"  0.3 )
( "minPRBoundaryInteriorHalo" "metal3"  0.4 )
```

The following rule is not defined correctly for mapping purposes:

```
( minEnclosure  ("prBoundary" "boundary") "metal1" 0.3 )
```

Assuming `(prBoundary boundary)` is specified in the mapping file, since `metal1` must be defined as the LPP; `(metal1 drawing)`, this rule will stay in the group/properties as user defined rules. It is not mapped to a constraint. The correct syntax is:

```
( minEnclosure ( "prBoundary" "boundary" ) ( "metal1" "drawing" ) 0.3 )
```

When correctly defined, the rule would map to the Virtuoso environment as:

```
( "minPRBoundaryInteriorHalo" "metal1"   0.3 )
```

Only the rules `minEnclosure` and `minExtension` are mapped. Rules other than `minEnclosure` and `minExtension` which use `prBoundary` will remain in CDBA (API) rule storage in the group/property as user defined rules and are not mapped to OpenAccess constraints. For example,

```
( minSpacing    metal1    prBoundary    0.6 )
```

Clusters

A cluster object is used to associate a group of instances and boundaries within a block. Each instance can only belong to one cluster. Clusters can be hierarchical.

For information about accessing cluster attributes, see Cluster Attributes and Cluster SKILL Functions in the *Virtuoso Design Environment SKILL Reference*.

Track Patterns

Track patterns are objects which provide guide lines for routing and define the position and number of routing tracks. Routing tracks are used by gridded routers to determine preferred locations and directions for routes. The line segments of track patterns are arranged in uniformly-spaced orthogonal patterns extending across a base array. Corresponds to OpenAccess `oaTrackPattern` class.

Technology File Requirements for Track Patterns

The following must be defined in your technology file in order to create track patterns.

```
layerDefinitions(
techPurposes(
(track 229 trk)
...
)
techDisplays(
(layerName track packetName t t t t nil)
...
)
techLayerPurposePriorites(
(layerName track)
...
)
 layerRules(
functions(
(layerName "metal" layerNumber)
...
)
```

To efficiently utilize the routing area, tracks should be created to consider the following design aspects:

Guidelines for Track Patterns

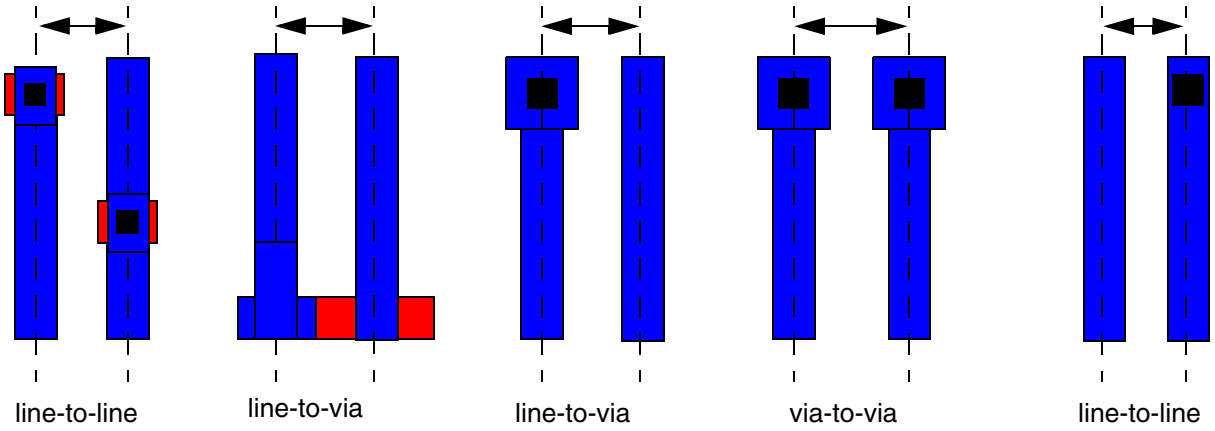
Tracks are drawn using the display attributes for LPP *routingLayerName* ***track***.

Tracks patterns are uniformly-spaced orthogonal patterns extending across the base array. The tracks that make up the track pattern, provide guidelines for laying interconnect routes. Gridded routers place the center-line of each route collinear with an unused segment of a routing track. The VLS-L and VLS-XL allows you to interactively create routes that you can snap to routing tracks. Routing along tracks ensures routes are created without violating minimum spacing rules and are designed to maximize the utilization of available routing area.

- Minimum Routing Pitch
- Via Stacking

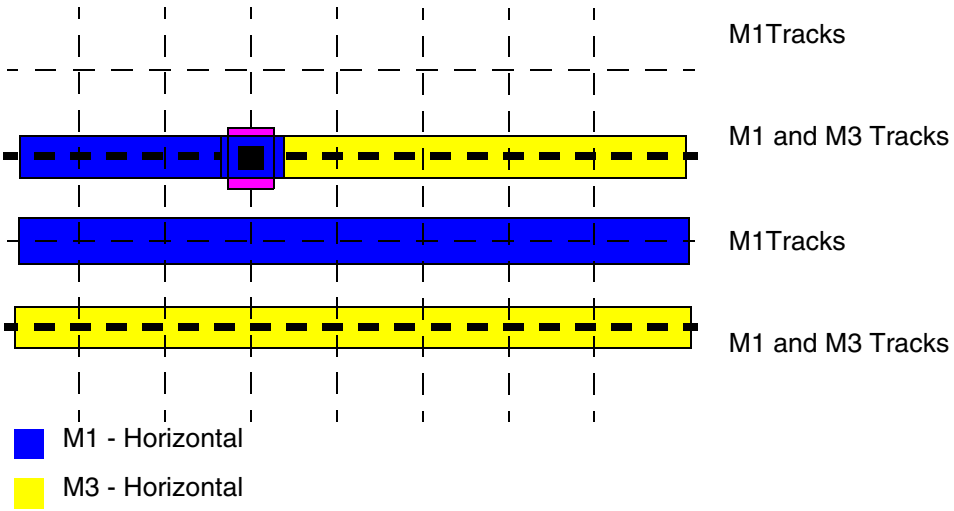
- Preferred and Non-Preferred Routing Directions

Minimum Routing Pitch



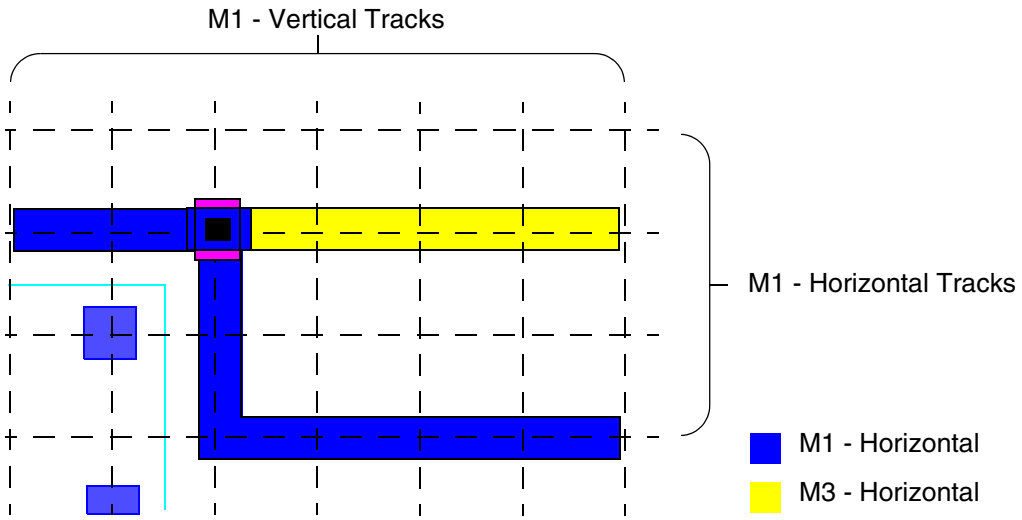
Track spacing is determined from the routing pitch. The type of vias used will determine the pitch. The most common routing pitch is line-to-line, because it allows the minimum pitch between routes with a cross type via. A hammer head type via will require a via-to-line routing pitch. A larger pitch does not efficiently utilize the routing area, where as smaller pitch may result in low utilization of the routing area because the router will not be able to place vias next to a route. For more information, see “Determining the Track Spacing” on page 371.

Via Stacking



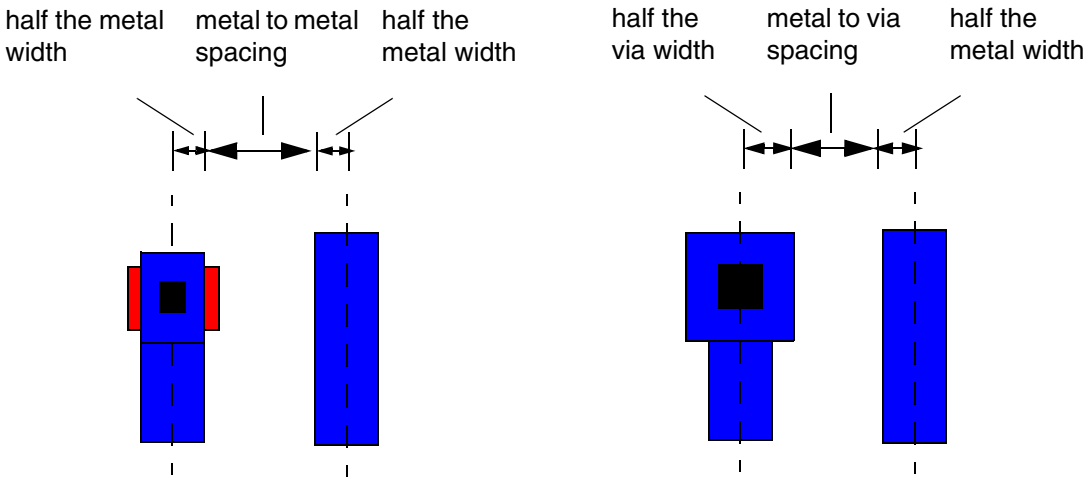
Routing layers using the same preferred direction can be placed on coincident tracks to allow for via stacking. If tracks can not be coincident, place the tracks at a ratio, for example, 1:2 or 2:3.

Preferred and Non-Preferred Routing Directions



Tracks are created in both the preferred and non-preferred routing direction to allow for X Y locations where routes can be jogged and vias and pins can be placed and connected. Points where a horizontal track on one layer crosses over a vertical track on another layer are via locations where a vias can be placed. Most routing is done on the tracks aligned in the preferred direction. Small-distance jogs use tracks in the non-preferred direction.

Determining the Track Spacing



Determine the line-to-via track spacing (routing pitch) for each routing layer by adding:

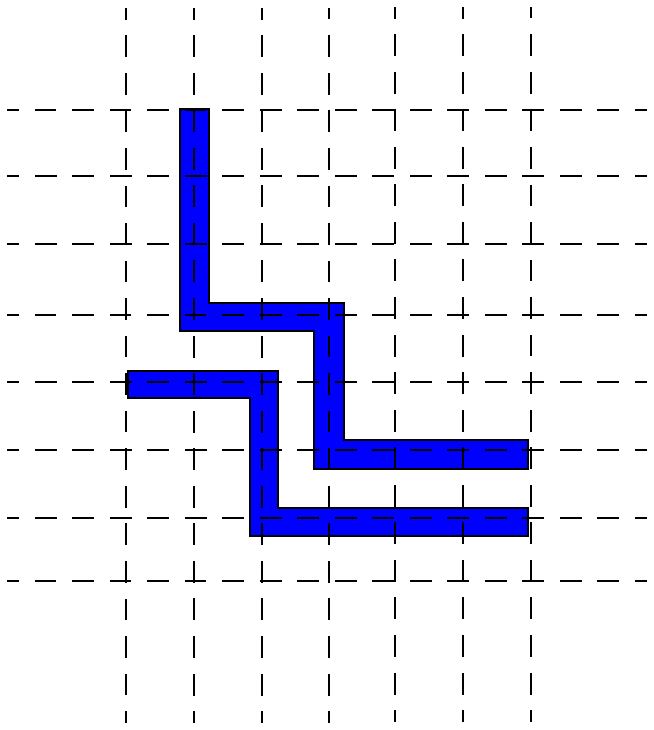
half the metal width + metal to metal spacing + half the width of the metal

or

half the via width + metal to via spacing + half the width of the metal

For information about different types of routing pitches, see “Minimum Routing Pitch” on page 369.

Determining the Number of Tracks



You must determine the number of tracks needed for the width and the height of the cellview. Calculate the number of horizontal tracks by dividing the height of the design area by the routing pitch of the metal. Determined the number of tracks needed in vertical direction using the same method. Create horizontal and vertical track patterns for each routing layer. When *Snap to Track* is on in the *Layout Editor Options* form, you are only allowed to route in areas where there are intersecting horizontal and vertical tracks.

For information about accessing track pattern attributes, see Track Pattern Attributes and Track Pattern SKILL Functions in the *Virtuoso Design Environment SKILL Reference* and Track Pattern C Functions in the *Cadence Integrator's Toolkit Database Reference*.

Markers

Markers are used to highlight design violations and the objects that are causing the violation which corresponds to OpenAccess `oaMarker` class.

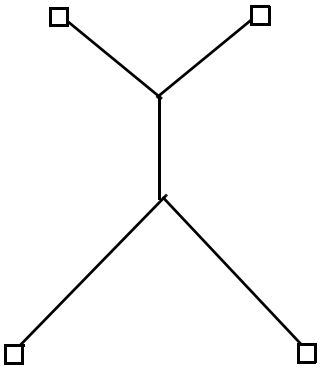
Several types of markers are supported in the Virtuoso Studio design environment.

- fatal error: tool generated error. Can not be signed off, as it will always result in a non-functioning design or a design that can not be manufactured. Such error are generated usually from the process end (design rule) where the far/foundry does not want to allow sign-off.
- critical error: tool generated error. Usually requires a design review to sign- off.
- signed off critical error: error has been reviewed by the design team, and has been signed-off.
- error: error markers.
- signed off error: error has been signed-off.
- warning: warning markers.
- acknowledged warning: warning has been acknowledged by the designer.
- info: tool generated information.
- annotation: user generated information.

For information about accessing marker attributes, see Marker Attributes and Marker SKILL Functions in the *Virtuoso Design Environment SKILL Reference* and Marker C Functions in the *Cadence Integrator's Toolkit Database Reference*.

Steiner

A steiner is used to represent a logical object that facilitates, or is used to minimize net length of branching wires.



A steiner is a logical point representing the branching start or end point or points of a route or routes. Steiners can be associated with a particular layer and net, and also have geometrical attributes.

For information about accessing Steiner attributes, see Steiner Attributes and Steiner SKILL Functions in the *Virtuoso Design Environment SKILL Reference* and Steiner C Functions in the *Cadence Integrator's Toolkit Database Reference*.

Layer Headers

In the Virtuoso environment, access to objects and shapes as they are grouped by layers is available at the API level through the objects, layerHeader and lppHeader.

A layerHeader objects holds information about a particular layer-purpose pair and the set of objects that use the layer purpose pair in a design database. The set of valid objects are; steiners, guides, blockages, and lppHeaders. The layerHeader provides direct access to these objects as they are grouped by layer.

A layerHeader is read only and can be bound or unbound. A bound layerHeader holds a valid technology database layer object. An unbound layerHeader only holds a layer number. layerHeader binding happens when layerHeader data is read in or a new layerHeader is created.

A layerHeader caches 3 union-ed bounding boxes for steiner, blockage and guide on the layer, one for each type of object. It also holds three lists of objects, one for each type for quick access.

The relationship of a layerHeader to objects is analogous to the relationship of an lppHeader to shapes. A layerHeader holds a collection of lppHeaders which contain the layer of the layerHeader. Therefore, a layerHeader provides indirect access to the shapes grouped by the lppHeaders.

For information about accessing cluster attributes, see Layer Header Attributes in the *Virtuoso Design Environment SKILL Reference* and Layer Header C Functions in the *Cadence Integrator's Toolkit Database Reference*.

CDB and Preview Conventions for Identifying Objects

Open Access Object Name	Conventions in CDB and Virtuoso Preview
route	In CDB and Virtuoso Preview, a route is equivalent to a collection of paths and via instances on connected LPPs.

row	In the CDB, a row is a rectangle on the LPP `row drawing`.
	
	In Virtuoso Preview, a row is represented by a rectangle on LPP (dbcRowLayer, dbcDrawingPurpose) and the following properties:
	
	`rowName`:
	`rowType`: `hRow`, `vRow`, `ioT`
	`placementClass`: site name
	`legalOrient`: site orientation
	`step`: value based on the width of the site and the orientation of the row.
	`alignment`: value of the distance between rows
	`rowOrient`: `R0`, `MXR90`

siteDef	In Virtuoso Preview, a rectangle on LPP (dbcPRBoundary, dbcDrawing) and the following properties: prCellType: site prCellClass: pad, core maskLayoutSubType: abstract symmetry: X, Y, R90
blockage	In CDB, a blockage is a rectangle or polygon on the `boundary` purpose, for example, `metal1 boundary`. If an LPP is not available, the system uses the following reserved LPP: `Cannotoccupy drawing` In Virtuoso Preview, a blockage is a rectangle on a layer with purpose boundary, stored in a dbGoup named defBlockages, with a string property of one of the following: defComponentBlockages defSlotsBlockages defFillsBlockages defPushdownBlockages defPlacementComponentBlockages defPlacementPushdownBlockages

Open Access Object Name	Conventions in CDB and Virtuoso Preview
prBoundary	In CDB, a boundary is a rectangle or polygon on the LPP `prBoundary drawing` (or `boundary`) In Virtuoso Preview, a rectangle or polygon on LPP (dbcPrBoundaryLayer, dbcBoundaryPurpose).
snapBoundary	In Virtuoso Preview, a snapBoundary is represented by a prBoundary bbox.
clusterBoundary	In Virtuoso Preview, a clusterBoundary is represented by region. It is parent-child relationship consisting of a rectangle and label. The rectangle is created on LPP (dbcPrBoundary, dbcDrawing) and has the following properties: regionType: normal regionName: region name placementClass: name objProperty: name-value pairs
cluster	In Virtuoso Preview, a cluster is a group of instances stored in a dbGroup with a string property named plcGroupType.
trackPattern	In Virtuoso Preview, track patterns are properties stored on the cellview that sets the global routing or GCell grid: the icXGGrid property specifies the location on the X axis and the icYGGrid property specified the location on the Y axis.
marker	In CDB and Virtuoso Preview, a warning marker is a rectangle on the LPP `marker warning`; an error is a rectangle on the LPP `marker error`
pinGroup	In CDB and Virtuoso Preview, subnets are used to model the *Strong*, *Weak*, and *Must Join* connectivity pin groups.

Controlling Display and Selection of Objects

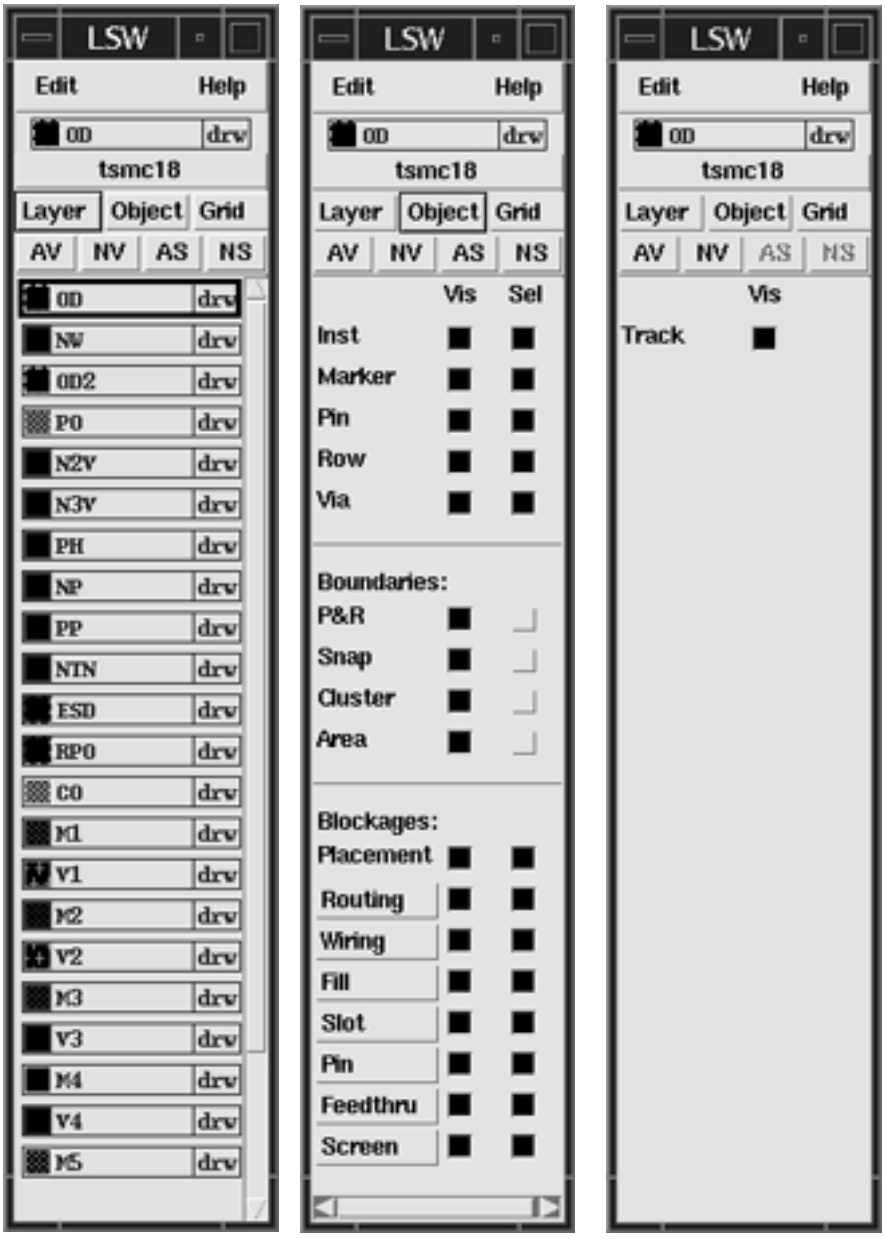
Required Technology File Layers and Purposes for OpenAccess Objects on page 379

Unplaced Instances on page 381

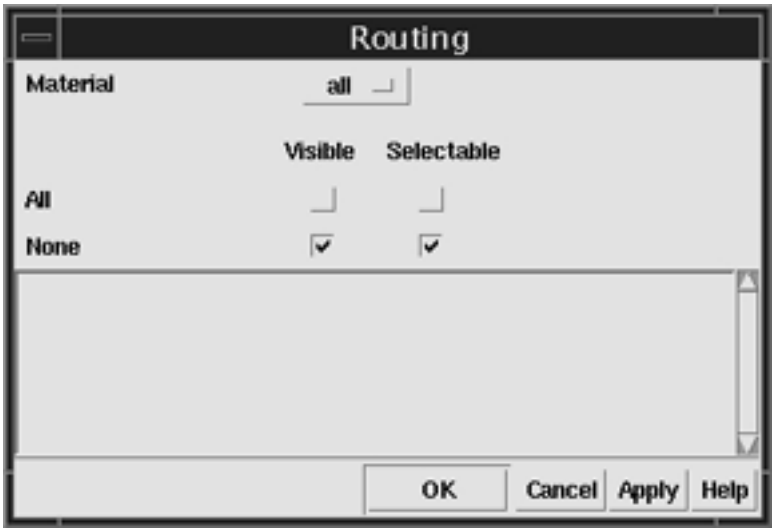
The *Objects* panel in the LSW lets you control visibility and selection of objects by turning on and off the relevant layer purpose pairs (LPPs). OpenAccess objects (marker, row, track

pattern, boundaries, and blockages) are assigned to an LPP to control selectability and visibility. Your technology file must contain the LPP's which support OpenAccess objects, and your display.drf file must contain packets for these LPP's.

The *Grid* panel lets you control the visibility of track patterns.



All blockage types, with the exception of placement, have forms which allow you to control the visibility and selectability on a layer by layer basis the blockages you create.



For more information, see Chapter 6, *The Layer Selection Window (LSW)* in the *Virtuoso Layout Suite L User Guide*.

Required Technology File Layers and Purposes for OpenAccess Objects

OpenAccess objects are assigned to layer purpose pairs (LPPs) to allow display and selection. The following table lists the layers and reserved purposes that allow you to control display and selection of OpenAccess objects. These layers and reserved purposes must be defined in the technology file. Corresponding packets are defined in the `default.drf` file located in the directory *`your_install_dir`*`/share/cdssetup/dfII`.

Important

The LPPs listed in the table are used for object display and selection and should be set to not valid in the LSW. This means the LPPs will not be available for shape drawing from the *Layer* panel of the LSW. Control selection and display of objects through the *Object* panel of the LSW.

The `default.drf` file contains predefined attributes such as colors, line styles, and stipples for LPPs and is further organized into a set of predefined packets. When starting the software, the `default.drf` is loaded and any the customized `display.drf` file are merged with the `default.drf`. Because the files are merged in sequence, files loaded later in the sequence can redefine display packets, colors, line styles, and stipples specified in files loaded earlier. Layer attributes you have defined in a customized `display.drf` loaded after the

`default.drf` will be respected. For example, the display packet for `snap boundary` is defined in a customized `display.drf` with a fill style of “solid”. This will overwrite the fill style of “outline” defined in the `default.drf`. In this case, all snap boundaries will be displayed with a solid fill obstructing the details of the cellview.

Object	Layer	System-Reserved Purpose
Standard Via	*viaLayer*	drawing
Row	Row	boundary
Boundary - prBoundary	prBoundary	boundary
Boundary - Cluster	group	boundary
Boundary - Area	border	boundary
Boundary - Snap	snap	boundary
Blockage - Placement	Cannotoccupy	boundary (see note below)
Blockage - Routing	*blockageLayer*	blockage (see note below)
Blockage - Wiring	*blockageLayer*	blockage (see note below)
Blockage - Fill	*blockageLayer*	blockage (see note below)
Blockage - Slot	*blockageLayer*	blockage (see note below)
Blockage - Pin	*blockageLayer*	blockage (see note below)
Blockage - Feedthru	*blockageLayer*	blockage (see note below)
Blockage - Screen	*blockageLayer*	blockage (see note below)
Placement Grid	*snapLayer*	grid
Manufacturing Grid	*routingLayer*	grid
Track pattern	*routingLayer*	track
Tack pattern - no layer	Unrouted	track
Marker - annotation	marker	annotation
Marker - info	marker	info
Marker - acknowledged warning	marker	ackWarning
Marker - warning	marker	warning
Marker - signed off error	marker	soError

Object	Layer	System-Reserved Purpose
Marker - error	marker	error
Marker - signed off critical error	marker	soCritical
Marker - critical error	marker	critical
Marker - fatal	marker	fatal

Note: To draw blockages depending on their type, you can add packet definitions to the `display.drf` file. You can define packets for `routing`, `fill`, `slot`, `pin`, `feedthru`, and `screen` type of blockages. The blockages you draw will be based on the attributes defined in these packets, if they exist. For naming the packet, prefix the blockage type with the packet name used by the blockage layer and an underscore, as in `M1Blockage_fill`. The following is an example of a definition of a new packet in the `display.drf` file:

```
( display M1Blockage_fill dots solid red red outlineStipple )
```

Unplaced Instances

Instances with the status of "unplaced" are not visible in the layout even though they are present in the design. The *Instance Statistics* section of the *Design - Summary* form displays the total number of unplaced instances.

See Placement Status on page 69 for more information.

Shapes

Paths with Extensions

On CDB you can specify extension values for a path of any style before setting the style as variable (`varExtendExtend`). In the Virtuoso environment, you need to set the style to `variable` before you specify values for extensions.

For OpenAccess, there is a potential loss of data if you change the type from variable to anything else. When changed from variable, the original settings, including extension information, are lost. If you then set the path style back to variable, you need to reset the extensions as well. This is not necessary in CDB.

The C function, `dbSetPathExt`, behaves differently for CDB and the Virtuoso environment:

- For CDB, the `dbSetPathExt` function always returns TRUE, no matter what the path style is. When the path style is not variable, specified extension data is applied to the path in the database, but ignored by applications.
- On the Virtuoso environment, `dbSetPathExt` returns FALSE when the path style is not variable, and specified extension data is not applied to the path in the database.

The C function, `dbGetPathExt`, behaves differently for CDB and the Virtuoso environment:

- For CDB, the `dbGetPathExt` function always returns the path extensions, no matter how the path style is set.
- In the Virtuoso environment, the `dbGetPathExt` function returns the path extension values when the style is variable but always returns zero for path extensions when the style is not variable.

Setting and querying the value of an extension differs for CDB and the Virtuoso Studio design environment:

- On CDB, you can set any value for extensions, independent of the style, and when you query an extension, the specified value is returned.
- In the Virtuoso environment, you cannot set a value for extensions unless the style is a variable. When you query an extension, the value returned is always zero (0.0) when the path style is not variable; when the style is variable, a query returns the extension value.

For example, for a path with the style `truncate`, setting and querying the beginning and ending extensions works as follows:

Set, then query	Results for CDB	Results for Virtuoso Studio design environment
path~>begExt = 2.5	2.5	nil
path~>begExt	2.5	0.0

Objects with Points

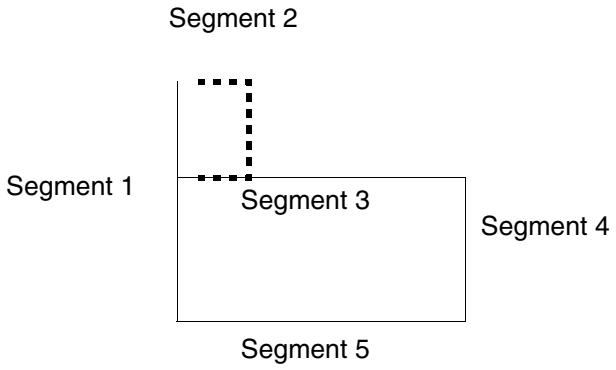
When a polygon or path is created, the Virtuoso environment removes any coincident and collinear points from the given point array to compress the amount of data used. CDB does not perform such checking. Therefore, data converted from CDB might have fewer points in OpenAccess. Although the bounding box will be the same, you need to review:

- Tests that compare or check the points or number of points
- Applications that rely on the ability to create collinear or coincident points

Retraced Segments in Polygons

Neither OpenAccess nor CDB allows self-intersecting polygons. However, for cases where a segment is created with points that are coincident with the points of an existing segment (when subsequent points retrace an existing segment), OpenAccess and CDB handle them as follows:

- OpenAccess drops the last two points when they retrace an existing segment. If fewer than three points remain, the CDBA procedural interface issues a warning saying that there were not enough points to form a polygon.



- CDB does not check for retraced segments in polygons.

For example, the following statement attempts to create a five-point polygon, with the fourth segment overlapping (retracing) the third segment:

```
dbCreatePolygon( cv "metal1" list( 1:2 4:2 4:7 5:8 4:7 ))
```

OpenAccess truncates the last two points (5:8 4:7), and creates a polygon from the remaining three points. CDB creates a polygon that looks like a line, with the fourth segment retracing the third segment.

The following statement attempts to create a four-point polygon, with the third segment overlapping the second segment:

```
dbCreatePolygon( cv "metal1" list( 1:2 4:2 4:7 4:2 ))
```

OpenAccess truncates points 4:7 and 4:2, and the CDBA procedural interface issues a warning saying there are not enough points to form a polygon. CDB creates a polygon that looks like a line, with the third segment overlapping the second segment.

Zero-Area Rectangles

OpenAccess does not allow the creation of a rectangle with zero size. CDB allows the creation of zero-area rectangles. Both databases issue warnings:

- OpenAccess: *WARNING* (`dbCreateRect`) Invalid bounding-box for a rectangle
- CDB: *WARNING* Rectangle being created has no area

For CDB, the GUI applications flag and correct zero-area rectangles before they get to the database. The only way to create zero-area rectangles in CDB is with `db` SKILL, `itk` or C functions. In OpenAccess, you cannot create zero-area rectangles.

Text Display Visibility

CDB and OpenAccess support different numbers of text display visibility attributes:

- CDB supports three independent text display visibility attributes: `visible`, `nameVisible`, and `valueVisible`, each of which can be `t` or `nil`.
- OpenAccess supports two independent visibility attributes:
  - `visible`
    a Boolean: `t` or `nil`
  - `oaTextDisplayFormat`
    allows three states: `nameVisible`, `valueVisible`, or `nameVisible&valueVisible`

As a result, CDB has eight possible visibility states, while OpenAccess has only six. Unfortunately, the existing API for setting individual text display visibility attributes can lead to transitions with particular sequences of flag settings that result in one of the following two states that are not supported in OpenAccess:

`(t n n)` and `(n n n)`

using state notation (`visible`, `nameVisible`, `valueVisible`).

In general, these are dubious states and are entered only as temporary states during a sequence of flag settings that are attempting to change the text display from one to another of the six OpenAccess-supported final states.

A new function

```
dbSetTextDisplayNameValueVisible( tdId nameVisble valueVisble )
```

has been added in C and SKILL to set both the `nameVisible` and `valueVisible` flags as a single operation, to prevent setting the text display visibility flags to the disallowed states. For a description of this function, see the dbSetTextDisplayNameValueVisible.

Warning messages are issued when an application tries to set the text display visibility flags to disallowed states under OpenAccess. Application developers need to know that illegal visibility states must be avoided, and the new warning messages help to identify these situations.

Bounding box for Net Expression Text Display Incorrect in OpenAccess

The size of the bounding box of a net expression text display for a terminal name attribute can be incorrect in OpenAccess. This is because OpenAccess does not evaluate net expressions from terminal name attributes.

CDB stores the terminal name and evaluates the net expression. OpenAccess stores the terminal name, but is unaware of the net expression. The size of net expression text display bounding boxes does not affect your design.

Unbound Text Displays

A cross-cellview text display object becomes unbound if it cannot be bound to its cross-cellview associate. In CDB, the unbound text display is deleted if the cellview containing the unbound text display is editable. In OpenAccess, the unbound text display is not deleted; instead, OpenAccess sets the text display text to

```
* Unbound *.
```

Arcs

Converting Start and Stop Angles for an Arc

In OpenAccess, the start and stop angles of an arc can be between 0 and 2PI. In CDB, start and stop angles can be between -PI and PI.

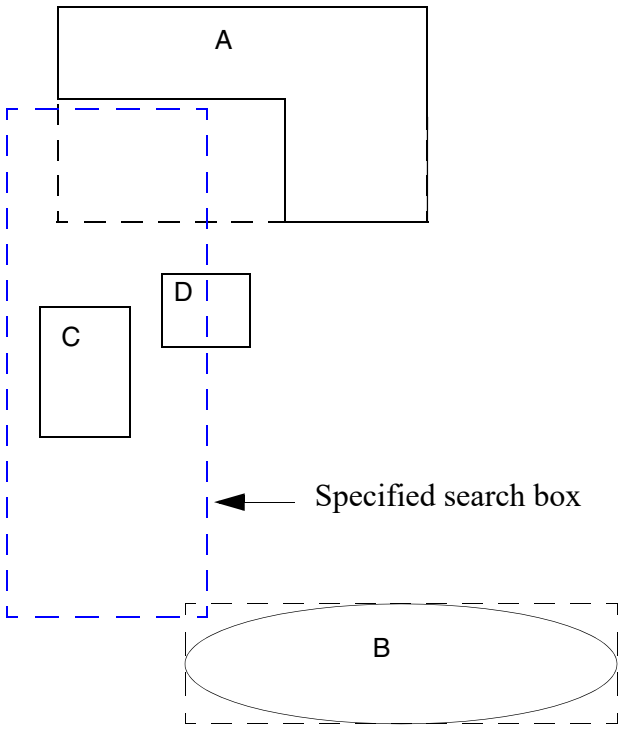
In the Virtuoso Studio design environment, querying an arc angle with `dbGetArcAngle` or `dbGetArcRec` returns angles between -PI and PI. Also when creating an arc with `dbCreateArc`, negative angles are accepted and the necessary adjustments are made.

However, if an application accesses the OpenAccess arc data directly, the application will need to make any necessary adjustments for start and stop angles that are between -PI and PI.

Selection of an Arcs

In the Virtuoso Studio design environment, the region query functionality has been changed to always support true overlaps of query regions for object geometries. See, Region Query Functions on page 67. Only objects whose geometry overlaps the specified region will be produced.

When selecting an arc, the arc must overlap the specified region in order to be selected.

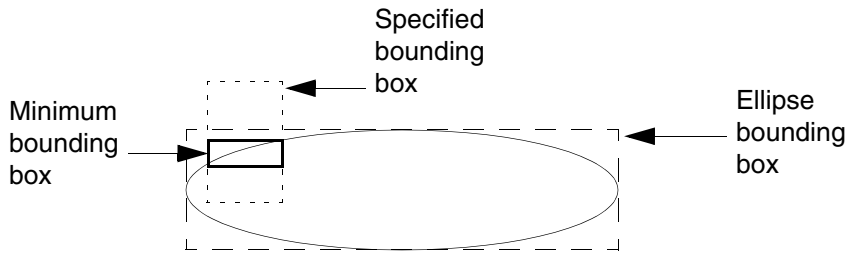


Figures produced are C, and D
Known as true overlaps

Querying an Arc for Its Bounding Box

When you query an arc for its bounding box, the values returned in CDB and OpenAccess will usually differ.

When you create an arc with the `dbCreateArc` function, one of the arguments you must specify is the bounding box of the arc. An arc is defined by the intersection of two bounding boxes: the bounding box of the ellipse of which the arc is a part, and the bounding box specified to delimit the arc. The specified delimiting bounding box need not be the minimum bounding box; it just needs to intersect with the ellipse bounding box to define the beginning and end of the arc. Therefore, the specified delimiting bounding box might be larger than the minimum bounding box needed to define the arc.



CDB saves both the ellipse and specified delimiting bounding boxes, and computes the start and stop angles from them. OpenAccess saves the ellipse bounding box and the start and stop angles, and computes the minimal bounding box that defines the arc.

In CDB, querying an arc for its bounding box, for example,

```
arc~>bBox
```

returns the value of the bounding box specified as an argument, which will probably be larger than the minimum bounding box required to define the arc.

In OpenAccess, querying an arc for its bounding box returns the exact bounding box of the arc, because OpenAccess calculates it from the ellipse bounding box and the start and stop angles.

Therefore, when you query an arc for its bounding box, the value returned by CDB and OpenAccess will probably be different.

Querying an Arc for Its Start and Stop Angles

The results of querying the start and stop angles of an arc might be slightly different for CDB and OpenAccess. This is because

- OpenAccess stores start and stop angles with double precision.
- CDB stores start and sweep angles with single precision. In CDB, the sweep angle is calculated by subtracting the start angle from the stop angle:

  ```
  Sweep Angle = ( stopAngle - startAngle )
  ```

So when you query the stop angle on CDB, the system calculates the result by adding the start angle and the stop angle.

Due to the difference between double and single precision, and the extra calculation for the stop angle performed by CDB, the following two query statements might return slightly different results for CDB and OpenAccess:

```
arc~>startAngle
arc~>stopAngle
```

Bounding Boxes of Labels and Text Displays

The bounding boxes of labels and text displays might be slightly different in OpenAccess and CDB. This is caused by a difference in how OpenAccess and CDB calculate their bounding boxes. OpenAccess rounds the values to the nearest database unit, where as CDB truncates them. The difference in the width and height of a label or text display could be off by 1 database unit (OpenAccess unit). Rounding to the nearest database unit is more accurate than truncating; CDB truncates.

If a label or test display is on the edge of a cellview, the difference described above might cause the cellview's bounding box to be different. The result would be that the bounding boxes of the instances of this cellview would also be different.

Also, when you query an arc for its bounding box, the values returned in OpenAccess and CDB differ slightly.

6

Migrating from PIPO to XStream

This chapter contains the following sections:

Overview

From the ICOA5.1.51 release PIPO has been replaced by XStream on OpenAccess. Both PIPO and XStream provide Stream Out capability to generate GDSII format data required for mask generation, and both provide a Stream In capability to restore that data into a Virtuoso environment database. However there are differences in the functionality and use model of PIPO and XStream. The following sections explain these differences in terms of command line usage, arguments and options, template file, and syntax.

One difference between PIPO and XStream is that you could, to some extent, use PIPO as a Virtuoso environment database archival mechanism. For example, using PIPO you can preserve Pcells and keep ROD data though these options are not sufficient to archive the complete design information, such as connectivity or user defined groups. XStream on the other hand does not allow you to preserve Pcells and ROD data. XStream does not duplicate the archival mechanism of PIPO in OpenAccess.

PIPO-XStream Template File Differences

As a PIPO user, you might be running PIPO Stream In - Stream Out from the command line or from customized scripts with the command line function embedded within. These scripts might use established PIPO template files which can be managed outside of the Virtuoso environment GUI.

Moving to XStream, some options and the template file syntax has changed, so you cannot directly use existing PIPO template files with XStream. For example, PIPO template file implements the SKILL list syntax, whereas the XStream template file uses a straight ASCII text, space, and line delimited format. Each line contains the option or argument, the value, and an optional comment or description.

For a Boolean argument, the value is not shown. A `nil` or an unasserted option is preceded by the comment character, `#`. A `t` or asserted option is just listed without a value as shown below:

PIPO	XStream Equivalent
`'ignoreBox "nil"`	`#ignoreBox`
`'ignoreBox "t"`	`ignoreBox`

For all text, lists, and other string values, the values can be specified with or without quotes.

Numeric values are specified without any special characters. In-line comments, if present, are preceded by the comment character, #.

Although most PIPO options have identical XStream equivalents, some may have the same functionality but not an identical option name. Some of the seemingly arbitrary option name changes in XStream have been done to maintain compatibility with the native OA Stream translator.

The following section explains the differences in the commands and the options between PIPO and XStream.

Stream In Template File Mapping

The following table maps the PIPO Stream In options with the corresponding XStream options with notes on differences in functionality and changes in the use model.

Stream In Template File Options

PIPO Option	XStream Option	Notes
runDir	runDir	Same options
inFile	strmFile	Equivalent options
primaryCell	topCell	Equivalent options
libName	library	Equivalent options
-	view	New option XStream allows you to specify the view name, default being `layout`.
techFileName	loadTechFile	Equivalent options
scale	-	Unsupported option In DFII on CDB, you could change the DBU/UU of a design at the library level. This is not permitted on OpenAccess and therefore not supported by XStream.

Stream In Template File Options

PIPO Option	XStream Option	Notes
errFile	logFile	In PIPO, the error file contains messages and summary information. Therefore the PIPO template option `errFile` is equivalent to using both the `logFile` and `summaryFile` options in XStream.
cellMapTable	cellMap	Equivalent options
layerTable	layerMap	Equivalent options
textFontTable	fontMap	Equivalent options
restorePin	pinAttNum	Equivalent options
propMapTable	propMap	Equivalent options
propSeparator	propSeparator	Same options
userSkillFile	userSkillFile	Same options The PIPO and XStream options are the same. In PIPO the following SKILL functions are supported: ■ piCellNameMap ■ piLayerMap ■ piTextMap ■ pipoErrShapesHandler ■ piPropMap In XStream all the above SKILL functions are supported for Stream In except `pipoErrShapesHandler`.

Stream In Template File Options

PIPO Option	XStream Option	Notes
-	objectMap	New option XStream provides an object map file to preserve non-maskable OpenAccess objects, such as boundaries and blockages. Object mapping file maps the object type and sub type to the Stream layer-datatype pair. This option is not applicable for PIPO.
-	viaMap	New option Vias are objects in OpenAccess. The via map file maps the Structure names from the Stream file to the corresponding viaDef names in the input technology library. The original vias can then be recreated during subsequent Stream Ins. This option is not applicable for PIPO.
checkPolygon	-	Unsupported option OA does not allow coincident or colinear points and removes them during creation of paths and polygons. XStream only reports incomplete or illegal polygons in the log file. Therefore this option is not supported by XStream.
snapToGrid	snapToGrid	Same options

Stream In Template File Options

PIPO Option	XStream Option	Notes
arrayToSimMosaic	arrayInstToScalar	Changed option By default on OA, mosaics are kept as OA arrays (AREF in SKILL). when you set `arrayInstToScalar` to t, XStream converts the arrays to scalar instances (SREF in SKILL). This is a reversal in behavior from `arrayToSimMosaic` in PIPO.
skipUndefinedLPP	skipUndefinedLPP	Same options

zeroPathToLine	ignoreZeroWidthPath	Changed option
		
		This PIPO option, `zeroPathToLine` can have the values <`lines/ignore`>. By default, during Stream In, XStream converts zero width paths into lines and they are ignored only when `ignoreZeroWidthPath` option is set to `t`.
		
		Therefore the mapping is as follows:
		
		`'zeroPathToLine "lines"`
		
		maps to
		
		`#ignoreZeroWidthPath`
		
		while
		
		`'zeroPathToLine "ignore"`
		
		maps to
		
		`ignoreZeroWidthPath`

caseSensitivity	case	Equivalent options
convertNode	translateNode	Equivalent options

Stream In Template File Options

PIPO Option	XStream Option	Notes
keepPcell	-	Unsupported option This PIPO option preserves parameterized cells (Pcells) by reading the SKILL files already created by Stream Out in the `KPDIR` directory located in the run directory. PIPO can be used to some extent for database archival whereas XStream is not meant to be used as a database archival mechanism.
mergeUndefinedPurposeToDrawing	mergeUndefPurposToDrawing	Equivalent options
reportPrecision	-	Unsupported option XStream does not perform this precision check and therefore this option is not supported by XStream.
ignoreBox	ignoreBoxes	Equivalent options
refLib	-	Unsupported option In PIPO this boolean option merely indicates whether to use reference libraries. If this flag is set to 't' then `refLibOrder` is searched for the list of reference libraries. XStream uses only one option `refLibList`. If `refLibList` is empty then it implies that no reference library is being used.
refLibOrder	refLibList	Equivalent options
noWriteExistCell	noOverwriteCell	Equivalent options

Stream In Template File Options

PIPO Option	XStream Option	Notes
runQuiet	-	Unsupported option XStream does not filter warning and information messages during translation. All error, warning, and information messages are written to the log file. Therefore this option is not supported by XStream.
NOUnmappingLayerWarning	-	Unsupported option XStream does not filter warning and information messages during translation. Therefore this option is not supported by XStream.
hierDepth	hierDepth	Same options
maxVertices	-	Unsupported option This option is not supported by XStream during Stream In though you can use this option during Stream Out.
rodDir	-	Unsupported option Using PIPO you can preserve Pcells and keep ROD data though these options are not sufficient to archive the complete design information. XStream does not allow you to preserve Pcells and ROD data as it is not intended to be used as an archival mechanism.
keepStreamCells	keepStreamCells	Same options
attachTechfileOfLib	attachTechFileOfLib	Equivalent options

Stream In Template File Options

PIPO Option	XStream Option	Notes
comprehensiveLog	-	Unsupported option This is a SKILL reporting mechanism which is not applicable on XStream. Therefore, this option is not supported by XStream.
appendDB	appendDB	Same option Use the *Append Existing Database* option to append data to an existing database. If the GDS file contains duplicate shapes or instances, they will be appended to the existing database.
genListHier	-	Unsupported option XStream writes hierarchical information in the summary file.
-	disableLocking	New option XStream provides the `disableLocking` option to disable file locking during Stream In. The disabling of file locking reduces the run time of Stream In especially if the designs contain a large number of cellviews in their hierarchy or library.

Stream In Template File Options

PIPO Option	XStream Option	Notes
-	summaryFile	New option XStream provides two files for messages and summary information whereas in PIPO the error file contains both messages and summary information. Therefore the PIPO template option `errFile` is equivalent to using both the `logFile` and `summaryFile` options in XStream.

Stream Out Template File Mapping

Stream Out Template File Options

PIPO Option	XStream Option	Notes
runDir	runDir	Same options
libName	library	Equivalent options
primaryCell	topCell	Equivalent options
viewName	view	Equivalent options
outFile	strmFile	Equivalent options
scale	-	Unsupported option In DFII on CDB, you could change the DBU/UU of a design at the library level. This is not permitted in OpenAccess and therefore not supported by XStream.

Stream Out Template File Options

PIPO Option	XStream Option	Notes
units	-	Unsupported option In DFII on CDB, you could change the DBU/UU of a design and set the units at the library level. This is not permitted in OpenAccess and therefore not supported by XStream.
hierDepth	hierDepth	Same options
convertToGeo	flattenPcells	Equivalent options This PIPO option flattens Pcells to geometry. XStream provides the *Flatten Pcells* option to flatten the Pcell instances in the designs.
maxVertices	maxVertices	Same options

Stream Out Template File Options

PIPO Option	XStream Option	Notes

refLib	refLibList	Equivalent options
		
		In PIPO this boolean option merely indicates whether to use reference libraries. XStream uses only one option `refLibList`. If `refLibList` is empty then it implies that no reference library is being used.
		
		Therefore the mapping is as follows:
		
		`'refLib t`
		
		maps to
		
		`refLibList "XST_CDS_LIB"`
		
		while
		
		`'refLib nil`
		
		maps to
		
		`refLibList ""`
		
		For more information on refLibList see the Design Translation Using XStream Translator chapter of the *Virtuoso Studio Design Environment Adoption Guide.*

libVersion	-	Unsupported option This PIPO option allows you to change the version number of the output GDSII Stream. Default: 5.0. In XStream the version number of the output GDSII Stream is always 5.0.

Stream Out Template File Options

PIPO Option	XStream Option	Notes
checkPolygon	-	Unsupported option OA does not allow coincident or colinear points and removes them during creation of paths and polygons. XStream only reports incomplete or illegal polygons in the log file. Therefore this option is not supported by XStream.
snapToGrid	snapToGrid	Same options
simMosaicToArray	arrayInstToScalar	Changed option By default on OA, mosaics are kept as OA arrays (AREF in SKILL). when you set `arrayInstToScalar` to t, XStream converts the arrays to scalar instances (SREF in SKILL). This is a reversal in behavior from `arrayToSimMosaic` in PIPO.
caseSensitivity	case	Equivalent options

Stream Out Template File Options

PIPO Option	XStream Option	Notes

lineToZeroPath	ignoreLines	Changed option
		In PIPO the template option `lineToZeroPath` can have the values `<path/ignore>`. By default, during stream out, XStream converts lines to paths and they are ignored only when `ignoreLines` option is set to `t`.
		Therefore the mapping is as follows:
		`'lineToZeroPath "path"`
		maps to
		`#ignoreLines`
		while
		`'lineToZeroPath "ignore"`
		maps to
		`ignoreLines`

convertDot	convertDot	Same options
rectToBox	rectToBox	Same options
convertPathToPoly	pathToPolygon	Equivalent options
keepPcell	-	Unsupported option XStream does not support preservation of Pcells. See the Overview section for details.
useParentXYforText	-	Unsupported option XStream does not do any data manipulation or data checking and therefore this option is not supported by XStream as yet.

Stream Out Template File Options

PIPO Option	XStream Option	Notes
reportPrecision	-	Unsupported option XStream does not perform this precision check and therefore this option is not supported by XStream.
runQuiet	-	Unsupported option XStream does not filter warning and information messages during translation. All error, warning, and information messages are written to the log file. Therefore this option is not supported by XStream.
errFile	logFile	Equivalent options In PIPO, the error file contains messages and summary information. Therefore the PIPO template option `errFile` is equivalent to using both the `logFile` and `summaryFile` options in XStream.
NOUnmappingLayerWarning	-	Unsupported option XStream does not filter warning and information messages during translation. Therefore this option is not supported by XStream.

Stream Out Template File Options

PIPO Option	XStream Option	Notes
techFileChoice	techLib	Changed option The XStream option `techLib` and the PIPO option `techFileChoice` are not equivalent. The `techFileChoice` option is to choose the technology file attached either to the input library or to each cellview that is traversed. A Virtuoso product can have any cellview attached to any technology file. The `techLib` option in XStream is for obtaining information about the layers and purposes used in the design. By default, the technology file of the source library is used to get information about the layers and purposes used in the design.
pcellSuffix	-	Unsupported option XStream does not support preservation of Pcells. See the Overview section for details.
respectGDSIILimits	respectGDSIINameLimit	Equivalent options
dumpPcellInfo	-	Unsupported option XStream does not support preservation of Pcells. See the Overview section for details.
cellMapTable	cellMap	Equivalent options
layerTable	layerMap	Equivalent options
textFontTable	fontMap	Equivalent options

Stream Out Template File Options

PIPO Option	XStream Option	Notes
convertPin	convertPin	Same options
pinInfo	pinAttNum	Equivalent options
pinTextMapTable	pinTextMap	Equivalent options
propMapTable	propMap	Equivalent options
propSeparator	-	Unsupported option By default, XStream uses a blank space as the property separator during Stream Out. You cannot specify a property separator during Stream Out though you can specify it during Stream In.
userSkillFile	userSkillFile	Equivalent options In PIPO the following SKILL functions are supported: ■ poCellNameMap ■ poParamCellNameMap ■ pipoErrShapesHandler ■ poLayerMap ■ textFontMap ■ poTextMap ■ poPropMap In XStream all the above SKILL functions are supported for Stream Out except `poParamCellNameMap` and `pipoErrShapesHandler`.

Stream Out Template File Options

PIPO Option	XStream Option	Notes
rodDir	-	Unsupported option This option is not supported by XStream. Using PIPO you can preserve Pcells and keep ROD data though these options are not sufficient to archive the complete design information. XStream does not allow you to preserve Pcells and ROD data as it is not intended to be used as an archival mechanism.
comprehensiveLog	-	Unsupported option This is a SKILL reporting mechanism which is not applicable on XStream. Therefore, this option is not supported by XStream.
genListHier	-	Unsupported option XStream writes hierarchical information in the summary file.
-	objectMap	New option This option is not supported by PIPO.
-	viaMap	New option This option is not supported by PIPO.

Stream Out Template File Options

PIPO Option	XStream Option	Notes
-	blockagePurpose	New option In XStream, you can specify the purpose on which blockages are to be translated. By default during Stream Out, blockages in the input OpenAccess database are translated as `BOUNDARY` elements in the output Stream file.
-	summaryFile	New option XStream provides two files for messages and summary information whereas in PIPO the error file contains both messages and summary information. Therefore the PIPO template option `errFile` is equivalent to using both the `logFile` and `summaryFile` options in XStream.

Using the PIPO-XStream Template Conversion Utility

If you have been using PIPO and want migrate to XStream you can convert your existing PIPO templates to XStream template file using the `pipo2Xstrm.pl` utility. This utility can be accessed from the `<install_directory>/tools/dfII/bin` directory. This utility creates an XStream template file from an existing PIPO template file.

The command line syntax to execute this `pipo2Xstrm.pl` is as follows:

```
pipo2Xstrm.pl {-in PIPOTemplateFileName
     [-out XStreamTemplateFileName]
     [-log TranslationLogFileName]
     [-overwrite]
     [-help] }
```

If the `pipo2Xstrm.pl` utility fails to run, then copy the utility from `<install_directory>/tools/dfII/bin` directory to you local bin directory and

modify the first line of code in the script by specifying the complete path from where you can use the perl executable:

```
#!/usr/bin/perl
```

change to:

```
#!<you_local_path_to_perl>
```

Note: You can exit the script using CTRL-C key combination.

This utility automatically determines whether the input PIPO template file is for Stream In or Stream Out and creates the corresponding XStream template. If you specify a log file name, the log file contains:

- XStream replacements for PIPO options
- XStream inserted options for those missing from the PIPO template
- additional options specific to XStream, and
- PIPO options not supported in XStream on OpenAccess.

Using the PIPO-XStream Utility

The following section contains the sample XStream files translated from PIPO and the corresponding log files that the `pipo2Xstrm.pl` utility generates.

Use the following commands to run the utility.

```
pipo2Xstrm.pl - in streamOut.il -out xstrmIn.templt -log xlatestrmIn.log
..............
WARNING: Handling of mosaics has changed - review documentation on
arrayInstToScalar
..............................
PIPO to XStream strmIn template translation complete.
Review template output. You may want to change file names, etc. from retained
values.
```

Following is a sample XStream stream in template file translated from PIPO

```
# XStream strmIn template translated from PIPO strmIn template.
# Source PIPO strmIn template file: streamIn.il
# Destination XSTREAM strmIn template file: xstrmIn.templt
# Wed Jul 20 20:46:51 2005
runDir      "."   # Run Directory
strmFile    "test.gds" # Input Stream File
```

```
topCell     ""    # Toplevel Cell to Translate
library     "strmLib"   # Destination Library
loadTechFile      ""    # ASCII Tech File
logFile     "PIPO.LOG" # Output Log File Name
hierDepth   32    # Hierarchical Depth to Translate to (0-32)
#snapToGrid       # (t/nil)
#arrayInstToScalar      # (t/nil)
case "lower"     # upper | lower | preserve
#ignoreZeroWidthPath    # (t/nil)
translateNode     # (t/nil)
#skipUndefinedLPP       # (t/nil)
#ignoreBoxes      # (t/nil)
#keepStreamCells # (t/nil)
attachTechFileOfLib     ""    # Tech Lib to Attach to Target Lib
#NoOverwriteCell # (t/nil)
cellMap     ""    # Input Cell Map File
layerMap    "layerMap" # Input Layer Map File
fontMap     ""    # Input Font Map File
pinAttNum   0     # Stream Attribute # (1-127) for Preserving Pins
propMap     ""    # Input Property Map File
propSeparator     ","   #
userSkillFile     ""    # Input User SKILL File
refLibList ""    # File Containing refLibList
#mergeUndefPurposToDrawing    # (t/nil)
viaMap      ""    # Input Via Map File
view "layout"    # Destination View Name
summaryFile       ""    # Output Summary File
#disableLocking   # (t/nil)
objectMap   ""    # Input Object Map File
```

Following is a sample of the corresponding pipo2Xstrm stream in translation log file

```
# Source PIPO strmIn template file: streamIn.il
# Destination XSTREAM strmIn template file: xstrmIn.templt
# Wed Jul 20 20:46:51 2005
runDir replaces runDir
strmFile replaces inFile
topCell replaces primaryCell
library replaces libName
loadTechFile replaces techfileName
scale is unsupported in XStream.
```

```
units is unsupported in XStream.
logFile replaces errFile
refLib is unsupported in XStream.
hierDepth replaces hierDepth
maxVertices is unsupported in XStream.
checkPolygon is unsupported in XStream.
snapToGrid replaces snapToGrid
WARNING: Handling of mosaics has changed - review documentation on
arrayInstToScalar
arrayInstToScalar replaces arrayToSimMosaic
case replaces caseSensitivity
ignoreZeroWidthPath replaces zeroPathToLine
translateNode replaces convertNode
keepPcell is unsupported in XStream.
skipUndefinedLPP replaces skipUndefinedLPP
ignoreBoxes replaces ignoreBox
reportPrecision is unsupported in XStream.
keepStreamCells replaces keepStreamCells
attachTechFileOfLib replaces attachTechfileOfLib
runQuiet is unsupported in XStream.
saveAtTheEnd is unsupported in XStream.
NoOverwriteCell replaces noWriteExistCell
NOUnmappingLayerWarning is unsupported in XStream.
genListHier is unsupported in XStream.
cellMap replaces cellMapTable
layerMap replaces layerTable
fontMap replaces textFontTable
pinAttNum replaces restorePin
propMap replaces propMapTable
propSeparator replaces propSeparator
userSkillFile replaces userSkillFile
rodDir is unsupported in XStream.
refLibList replaces refLibOrder
comprehensiveLog is unsupported in XStream.
Adding missing PIPO option equivalents with default values.
Inserting mergeUndefPurposToDrawing with default value 'nil'
Adding new XStream options with default values.
New option 'viaMap' with value '""'
New option 'view' with value '"layout"'
New option 'summaryFile' with value '""'
New option 'disableLocking' with value 'nil'
```

```
New option 'objectMap' with value '""'
PIPO to XStream strmIn template translation complete.
Review template output. You may want to change file names, etc. from retained
values.
```

Following is a sample XStream stream out template file translated from PIPO

```
# XStream strmOut template translated from PIPO strmOut template.
# Source PIPO strmOut template file: streamOut.il
# Destination XSTREAM strmOut template file: xstrmOut.templt
# Wed Jul 20 21:13:45 2005
runDir      "."   # Run Directory
library     "LIB"       # Input Library
topCell     ""    # Toplevel Cell to Translate
view "layout"    # Toplevel Cell View Name
strmFile    "test.gds" # Output Stream File
hierDepth   32    # Hierarchical Depth to Translate to (0-32)
#flattenPcells    # (t/nil)
maxVertices       200   # Limit of Vertices Allowed
#refLibList       # Name of File Containing refLibList
#snapToGrid       # (t/nil)
#arrayInstToScalar      # (t/nil)
case "upper"     # upper | lower | preserve
#ignoreLines      # (t/nil)
convertDot "node"      # node | polygon | ignore
#rectToBox # (t/nil)
#pathToPoly       # (t/nil)
logFile     "PIPO.LOG" # Output Log File Name
#respectGDSIINameLimit # (t/nil)
cellMap     ""    # Input Cell Map File
layerMap    "layerMap" # Input Layer Map File
fontMap     ""    # Input Font Map File
convertPin "geometry" # geometry | text | geometryAndTxt | ignore
pinAttNum   0     # Stream Attribute # (1-127) for Preserving Pins
pinTextMap ""    # Input Pin Text Map File
propMap     ""    # Input Property Map File
userSkillFile     ""    # Input User SKILL File
viaMap      ""    # Input Via Map File
summaryFile       ""    # Output Summary File
blockagePurpose   "boundary" # Purpose Name for Blockages
techLib     ""    # Technology Library
```

```
objectMap   ""    # Input Object Map File
```

Following is a sample of the corresponding pipo2Xstrm stream in translation log file

```
# Source PIPO strmOut template file: streamOut.il
# Destination XSTREAM strmOut template file: xstrmOut.templt
# Wed Jul 20 21:13:45 2005
runDir replaces runDir
library replaces libName
topCell replaces primaryCell
view replaces viewName
strmFile replaces outFile
scale is unsupported in XStream.
units is unsupported in XStream.
hierDepth replaces hierDepth
flattenPcells replaces convertToGeo
maxVertices replaces maxVertices
refLibList replaces refLib
libVersion is unsupported in XStream.
checkPolygon is unsupported in XStream.
snapToGrid replaces snapToGrid
WARNING: Handling of mosaics has changed - review documentation on
arrayInstToScalar
arrayInstToScalar replaces simMosaicToArray
case replaces caseSensitivity
ignoreLines replaces lineToZeroPath
convertDot replaces convertDot
rectToBox replaces rectToBox
pathToPoly replaces convertPathToPoly
keepPcell is unsupported in XStream.
useParentXYforText is unsupported in XStream.
reportPrecision is unsupported in XStream.
runQuiet is unsupported in XStream.
logFile replaces errFile
NOUnmappingLayerWarning is unsupported in XStream.
techFileChoice is unsupported in XStream.
pcellSuffix is unsupported in XStream.
genListHier is unsupported in XStream.
respectGDSIINameLimit replaces respectGDSIILimits
dumpPcellInfo is unsupported in XStream.
cellMap replaces cellMapTable
```

```
layerMap replaces layerTable
fontMap replaces textFontTable
convertPin replaces convertPin
pinAttNum replaces pinInfo
pinTextMap replaces pinTextMapTable
propMap replaces propMapTable
propSeparator is unsupported in XStream.
userSkillFile replaces userSkillFile
rodDir is unsupported in XStream.
comprehensiveLog is unsupported in XStream.
genListHier is unsupported in XStream.
Adding missing PIPO option equivalents with default values.
Adding new XStream options with default values.
New option 'viaMap' with value '""'
New option 'summaryFile' with value '""'
New option 'blockagePurpose' with value '"boundary"'
New option 'techLib' with value '""'
New option 'objectMap' with value '""'
PIPO to XStream strmOut template translation complete.
Review template output. You may want to change file names, etc. from retained
values.
```

A

prRules Conversion Messages

Understanding prRules Conversion Messages

This section describes the important information, warning, and error messages you might see in the CIW or in a dialog box when

- You load an ASCII technology file, and the system checks the content of the file and reports load information and errors.
- You load a technology database that was converted to OpenAccess prior to the ICOA 5.0.33 USR1 release, and the system automatically reconverts the database and reports conversion information and errors.

The messages are listed alphabetically. Some messages begin with variables to represent information specific to the message, such as a function name.; these messages are listed first.

Note: This construct has been deprecated and will be removed in an upcoming release. Cadence recommends that you remove it from your technology files.

Message	Problem and Solution
Function name: Cannot find viaDef *viaDef1*.	**Problem:** viaDef is not found in the technology database. **Solution:** Make sure the name of the `viaDef` corresponds to an existing `via` or `viaDef`.
Function name: Cannot open cdsVia lib/cell/view.	**Problem:** The tool tried to open the cellview that corresponds to the `cdsVia`, but failed. **Solution:** Make sure the `cdsVia` cellview is properly created.

Message	Problem and Solution
Function name: Could not create viaspec.	**Problem:** Could not create a viaSpec for the prViaRule. **Solution:** Check the minimum width (`wMin`) and maximum width (`wMax`) parameters for the layers to see if they are in the correct format.
Function name: direction *direction1* is not stored on layer *layer1* in the database; preferred direction *direction2* is used.	**Problem:** The direction specified in `prViaRule` and `prGenViaRule` is compared with the direction stored in the technology database. When they differ, this warning is issued. **Solution:** Verify whether there is a conflicting definition for direction in the technology file or technology database.
Function name: Failed to get the viaParams from the prGenViaRule parameters.	**Problem:** Could not get the stdViaDef parameters from the prViaRule values. **Solution:** Check the parameters for prViaRule to see if they are in the correct format.
Function name: Failed to update the parameters of viaDef *viaDef1*.	**Problem:** When the function tried to update the existing viaDef parameters with the prViaRule values, the operation failed. **Solution:** Check the parameters for the prViaRule to see if they are in the correct format.
Function name: Failed to update the parameters of viaDef *viaSpec1*.	**Problem:** When the function tried to update the existing viaSpec parameters with the prViaRule or prGenViaRule values, the operation failed. **Solution:** Check the parameters for the prViaRule to see if they are in the correct format.
Function name: Invalid spacing type for layer.	**Problem:** The data type for the spacing of the routing layer in the prNonDefaultRule must be floating point. **Solution:** Check the data type for the spacing of the routing layer.

Message	Problem and Solution
Function name: Invalid stackViaParams for isStackable.	**Problem:** The data type for the stackable value of the via layers in the prNonDefaultRule must be Boolean. **Solution:** Check the data type for the stackable value of the via layers and make sure it is Boolean.
Function name: Invalid via name *via1*.	**Problem:** The specified viaDef name does not represent a valid prVia. **Solution:** Check the name of the viaDef and make sure it is a valid prVia.
Function name: Invalid width type for layer.	**Problem:** The data type for the width of the routing layer in the prNonDefaultRule must be floating point. **Solution:** Check the data type for the width of the routing layer.
Function name: Layer *layer1* is not a valid routing layer.	**Problem:** A valid `prRouting` layer is a physical layer that has a mask number and a function that is not of the type “other” (unknown). **Solution:** Make sure that the layer has all required attributes set.
Function name: Not a valid Enclosure or Direction value.	**Problem:** The data type for the layer direction or overHang is not correct. **Solution:** Check the format of the layer direction and overHang.
Function name: Not a valid lowerPt value.	**Problem:** The lowerPt is not valid because it has either an incorrect data type or an incorrect length. **Solution:** Check the format of the lowerPt in the prGenViaRule.

Message	Problem and Solution
Function name: Not a valid xPitch and yPitch value.	**Problem:** The xPitch or yPitch does not have a valid data type. **Solution:** Check the data type for the xPitch and yPitch parameters in the prGenViaRule.
Function name: metalOverHang and overHang are not stored in the database.	**Problem:** The rule names `metalOverHang` and `overHang` are derived from layer enclosure and direction. OpenAccess does not provide direct attributes. **Solution:** No action required.
Function name: The stackable via value of *value1* and *value2* is not compatible with the existing value.	**Problem:** The value for spacing or the stackable via in the prNonDefaultRule is different than the value in the physicalRules section or the value in the prStackVias section. **Solution:** Check the stackable and minSpacing values defined for the spacingRules in the physicalRules and prStackVias sections. Make sure the values are the same as in the prNonDefaultRule section.
Function name: viaDef creation for ruleContact device only supports two adjacent routing layers.	**Problem:** `ruleContact` device defines more than two layers and more than one via layer. Only `layer0`, `layer1`, and the `via10` layer are used to create a `viaDef`. **Solution:** No action required.

Message	Problem and Solution
Function name: viaDef *viaDef1* is not a valid prVia.	**Problem:** A valid `prVia` is defined as: ■ Contained in the technology library. ■ Has a cell name and view name that correspond to a valid cellview. ■ At least one of the layers in the cellview is a valid routing layer, which implies that the `LEFDefaultRouteSpec` exists. The other layer could be either a valid routing layer or a valid masterslice layer. **Solution:** Verify whether the `viaDef` corresponds to a via that satisfies the conditions above.
Convert prRules: No device found in the technology database.	**Problem:** There is no device (via) found in the technology database. Conversion of **prRules** to **routeSpec** is likely to fail because cut layers are usually found in vias. **Solution:** Your technology file or technology database might be incomplete. Update the file or database to include via definitions or define cut layers.
Convert prRules: No routing layer found in the technology database.	**Problem:** `prRules` are defined in the technology database, but routing layers are not specified. **Solution:** Routing layers are required for `prRule` conversion. You need to add the routing layers to the `prRoutingLayer` rule.
Convert prRules: No valid cut layer found between routing layer *layer1* and *layer2*.	**Problem:** There must be a cut (via) layer between two routing layers. The tool could not find a cut layer from either the existing devices or the `viaLayer` rule. **Solution:** Verify whether the routing layers are properly defined and whether a via exists that contains both routing layers. Or, you can create a `viaLayer` rule for the two routing layers.

Message	Problem and Solution
Convert prRules: prRule layer mask numbers must be fully specified.	**Problem:** In the technology database, some layers have mask numbers assigned, while other layers do not. For layers with no mask number, the tool could not automatically assign a mask number. **Solution:** Assign mask numbers for all routing layers, masterslice layers, and cut layers.
Not valid via layer pair.	**Problem:** Could not find the stackable via layers specified by the prNonDefaultRule. **Solution:** Check the stackable via layers specified in the prNonDefaultRule.