Contents

Preface ................................................................................................................. 13
Related Documents .............................................................................................. 13
Typographic and Syntax Conventions .................................................................... 14
   SKILL Syntax Examples ..................................................................................... 15

1
Overview of the Translators ..................................................................................... 17
Introduction ............................................................................................................ 18
Understanding the Translators ................................................................................. 18
   General Translators ............................................................................................ 19
   Netlist Translators .............................................................................................. 19
   Physical Translators ............................................................................................ 19
How to Start the Translators .................................................................................... 20

2
Preparing to Use Stream and CIF Translators ......................................................... 23
Overview ............................................................................................................... 24
Scaling Your Design ............................................................................................... 24
   Scaling Process .................................................................................................. 24
   Scaling Example ................................................................................................. 25
Using Map Files When Importing ........................................................................... 26
   Cell Name Mapping File .................................................................................... 27
   Layer Name Mapping File .................................................................................. 28
   Text Font Mapping File for Stream In ............................................................... 32
   Text Font Mapping File for CIF ......................................................................... 34
   Property Mapping File for Stream In ................................................................. 35
Using Map Files when Exporting ............................................................................ 36
   Cell Name Mapping File for Stream Out ........................................................... 37
   Layer Name Mapping File for Stream Out ........................................................ 37
   Text Font Mapping File for Stream Out ............................................................ 41
## 3 Translating Stream Files

- **Overview** .................................................. 48
- **Translating Reference Libraries Using Stream** ............... 48
- **Retaining Reference Library Structure in Stream Files** .... 49
  - Translating a Stream File with Reference Information .......... 51
  - Translating a Stream File with Complete Structures .......... 52
- **Moving Stream Files between Tape and Disk** ................ 52
  - Recreating Relative Object Design Data during Translation Using Non-Cadence Tools ........................................ 53
- **Using Import – Stream** ...................................... 53
  - **Import – Stream Forms** ...................................... 57
  - Translating Compressed Stream File .............................. 76
  - Starting Import – Stream from a Form .......................... 76
  - Starting Import – Stream from the Command Line .............. 77
  - Stream In GUI and Template File Options ....................... 77
  - Preparing a Template File and .cdsinit File for Stream In .... 79
- **How Import – Stream Translates Data** ......................... 82
- **Import – Stream Output Files** ................................ 84
  - **Error File** .................................................. 86
  - **ASCII Dump File** ............................................ 90
- **Using Export – Stream** ........................................ 91
  - **Export – Stream Forms** ...................................... 94
  - Starting Stream Out from a Form ................................ 120
  - Starting Export – Stream from the Command Line .............. 121
  - Stream Out GUI and Template File Options .................... 122
  - Preparing a Template File and a .cdsinit File for Stream Out ........................................ 124
- **How Export – Stream Translates Data** ......................... 127
  - **Export – Stream Data Types** ................................ 128
  - Interpreting the Units Data .................................... 128
  - **Export – Stream Output Files** ............................... 130
4 Translating CIF Files

Overview
Using Import – CIF
Import – CIF
Preparing a Template File and .cdsinit File
How Import – CIF Translates Data
Import – CIF Output Files
Error File
Template File
ASCII Dump File
Exporting with CIF
Export – CIF
Preparing a Template File and .cdsinit File
How Export – CIF Translates Data
Export – CIF Output Files
Error File
Template File
ASCII Dump File

5 CDL Translators

Overview
Using CDL In
6
Using SKILL to Customize Stream and CIF Files

Overview

Customizing the Graphical User Interface of Translators

Customizing Input Files

Customizing Export Files
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>poCellNameMap</td>
<td>280</td>
</tr>
<tr>
<td>poCifLayerMap</td>
<td>281</td>
</tr>
<tr>
<td>textFontMap</td>
<td>281</td>
</tr>
<tr>
<td>pipoErrShapesHandler</td>
<td>283</td>
</tr>
<tr>
<td>poParamCellNameMap</td>
<td>285</td>
</tr>
<tr>
<td>poTextMap</td>
<td>287</td>
</tr>
</tbody>
</table>

7 Translating LEF and DEF Files

Overview .................................................. 290

LEF and DEF Translation Commands ......................... 290
  DEFINE Statements .................................. 291
  ALIAS Statements .................................. 291
  Version Settings .................................. 293

LEF/DEF Translation Process ............................. 294
  Invoking LEF/DEF Translator ....................... 294
  Running LEF In from the GUI ....................... 294
  Running LEF In from the Command Line ............ 299
  Translating a Compressed LEF File ................. 301
  Running LEF Out from the GUI ..................... 302
  Running LEF Out from the Command Line .......... 307
  Running DEF In from the GUI ....................... 310
  Running DEF In from the Command Line .......... 314
  Translating a Compressed DEF File ................. 315
  Running DEF Out from the GUI ..................... 316
  Running DEF Out from the Command Line .......... 320

LEF Data Map ............................................. 323
  VERSION Statement ................................ 324
  UNITS Section ...................................... 324
  LAYER Section ...................................... 327
  MAXVIASTACK Statement ............................ 354
  PROPERTYDEFINITIONS Section ..................... 355
  VIAS Section ...................................... 355
  VIARULE Section .................................. 362
  NONDEFAULTRULE Section ......................... 366
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPACING Section</td>
<td>369</td>
</tr>
<tr>
<td>SITE Section</td>
<td>369</td>
</tr>
<tr>
<td>ARRAYS Section</td>
<td>370</td>
</tr>
<tr>
<td>MACRO Section</td>
<td>372</td>
</tr>
<tr>
<td>PIN Section</td>
<td>376</td>
</tr>
<tr>
<td>OBS Section</td>
<td>384</td>
</tr>
<tr>
<td>ANTENNA Statements</td>
<td>385</td>
</tr>
<tr>
<td>MANUFACTURINGGRID Statement</td>
<td>385</td>
</tr>
<tr>
<td>USEMINSPACING Statement</td>
<td>386</td>
</tr>
<tr>
<td>CLEARANCEMEASURE Statement</td>
<td>387</td>
</tr>
<tr>
<td>DEF Data Map</td>
<td>387</td>
</tr>
<tr>
<td>VERSION Section</td>
<td>388</td>
</tr>
<tr>
<td>PROPERTYDEFINITIONS Section</td>
<td>390</td>
</tr>
<tr>
<td>VIAS Section</td>
<td>390</td>
</tr>
<tr>
<td>REGIONS Section</td>
<td>391</td>
</tr>
<tr>
<td>COMPONENTS Section</td>
<td>392</td>
</tr>
<tr>
<td>PINS Section</td>
<td>394</td>
</tr>
<tr>
<td>IOTimings Section</td>
<td>399</td>
</tr>
<tr>
<td>NETS Section</td>
<td>401</td>
</tr>
<tr>
<td>SPECIALNETS Section</td>
<td>406</td>
</tr>
<tr>
<td>GROUPS Section</td>
<td>412</td>
</tr>
<tr>
<td>CONSTRAINTS Section</td>
<td>413</td>
</tr>
<tr>
<td>BLOCKAGES Section</td>
<td>414</td>
</tr>
<tr>
<td>SLOTS Section</td>
<td>416</td>
</tr>
<tr>
<td>FILLS Section</td>
<td>417</td>
</tr>
<tr>
<td>SCANCHAINS Section</td>
<td>418</td>
</tr>
<tr>
<td>SKILL Functions</td>
<td>419</td>
</tr>
<tr>
<td>ldtrDefRead</td>
<td>419</td>
</tr>
<tr>
<td>ldtrDefWrite</td>
<td>421</td>
</tr>
<tr>
<td>ldtrGenLefoutList</td>
<td>423</td>
</tr>
<tr>
<td>ldtrLefRead</td>
<td>425</td>
</tr>
<tr>
<td>ldtrLefWrite</td>
<td>427</td>
</tr>
</tbody>
</table>
A
Stream Format ................................................................. 429
  Overview ................................................................. 430
  About GDSII ............................................................... 430
    History ................................................................. 430
    GDSII Format .......................................................... 430
  Stream File Description ................................................ 431
    Stream Data Types .................................................... 431
    Stream Records ........................................................ 434
  Differences in Stream Format Versions ............................... 449
  Stream Syntax ............................................................ 449
  Example of a Stream Format File ..................................... 451
  How to read, write, and print Stream dates to avoid Y2K problems ........................................... 460
    Code Examples .......................................................... 460

B
PIPO Messages ............................................................... 463
  Introduction ............................................................... 463
  Invoking Extended Help ................................................ 465
  Error Messages and Warnings ......................................... 466

C
CDB Abstract Cellview Properties for Exporting LEF/DEF ........ 579
  Introduction ............................................................... 579
    The maskLayoutSubType Property .................................. 579
    The prCellType and prCellClass Property ......................... 580
    The function Property ............................................... 581
    The placementClass Property ....................................... 581
  CDB Abstract Cellview Property for Exporting DEF ............... 582

D
CDL and SPICE ............................................................... 583
  How CDL Differs from SPICE ........................................... 583
Design Data Translator's Reference

Sample SPICE File .......................................................... 584

Index ................................................................. 585
Preface

This manual assumes that you are familiar with the development and design of integrated circuits. It also assumes you know one of the following languages: GDSII Stream, CIF and CDL. It contains reference information about the following Cadence® products:

- **Virtuoso® Stream In**, which translates files from GDSII Stream format into the Design Framework II™ database format.
- **Virtuoso® Stream Out**, which translates files from Design Framework II format into the GDSII Stream format.
- **Virtuoso® CIF In**, which translates files from Caltech Interchange Format into the Design Framework II database format.
- **Virtuoso® CIF Out**, which translates files from Design Framework II format into the Caltech Interchange Format.
- **Virtuoso® CDL In**, which translates netlists in the Circuit Description Language (CDL) to the Design Framework II™ CDB database format.
- **Virtuoso® CDL Out**, which translates files from Design Framework II format into the Circuit Description Language format.
- SKILL functions to customize Stream and CIF files
- LEF and DEF mapping with the dfII environment

The preface discusses the following:

- Related Documents on page 13
- Typographic and Syntax Conventions on page 14

Related Documents

The translators are often used with other Cadence products, or require knowledge of special language (SKILL, TLF, DEF, Verilog). The following documents give you more information about these tools and languages.

If you want to use the Cadence Layout Editor to enter your design, you should read **Virtuoso Layout Editor User Guide**.
Circuit Description Language (CDL) format is a subset of SPICE format. CDL uses node names as well as node numbers. For more information about CDL syntax, see the LOGLVS chapter of the *Dracula Reference*.

### Typographic and Syntax Conventions

This list describes the SKILL syntax conventions used for the SKILL functions in this manual.

**literal**

Non-italic words indicate keywords that you must enter literally. These keywords represent command (function, routine) or option names.

**t_argument**

Words in italics indicate user-defined arguments for which you must substitute a name or a value. Names are case sensitive. The characters before the underscore (\_) in the word indicate the data types that this argument can take. Do not type the datatype (t\_) before your arguments.

The following is a list of valid data types:

- **d** = dbobject A database object name
- **t** = string An alphabetic letter or letters
- **s** = symbol A symbol name
- **p** = I/O port An input or output port name
- **l** = list A list of names
- **x** = integer A numeric integer
- **f** = float A floating point number
- **g** = general Any of the above data types

| Vertical bars (OR-bars) separate possible choices for a single argument. They take precedence over any other character.

[ ] Brackets denote optional arguments. When used with OR-bars, they enclose a list of choices from which you can choose one.
SKILL Syntax Examples

The following examples show typical syntax characters used in SKILL.

Example 1

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

Example 1 illustrates the following syntax characters.

- **list**
  - Plain type indicates words that you must enter literally. There is no space allowed between the literal function name and the left parenthesis. For example,
    ```
    list( is the correct syntax and not list ( g_arg1
    ```
  - Words in italics indicate arguments for which you must substitute a name or a value.
  - Parentheses separate the arguments from the names of functions.

- **{ }**
  - Braces are used with OR-bars and enclose a list of choices.
  - You must choose one argument from the list.

- **...**
  - Three dots ( . . . ) indicate that you can repeat the previous argument. If you use them with brackets, you can specify zero or more arguments. If they are used without brackets, you must specify at least one argument, but you can specify more.

- **[argument]**
  - You can specify zero or more.

- **,...**
  - A comma and three dots together indicate that if you specify more than one argument, you must separate those arguments by commas.

- **=>**
  - A right arrow precedes the possible values that a SKILL function can return.

- **/**
  - A slash separates the possible values that can be returned by a SKILL function.
Example 2

Example 2 illustrates two additional syntax characters.

| Vertical bars separate a choice of required options.

/ Slashes separate possible return values.
Overview of the Translators

This chapter describes the following:

- Introduction on page 18
- Understanding the Translators on page 18
- How to Start the Translators on page 20
Introduction

The Cadence® Design Framework II environment is the foundation on which most Cadence tools are built. The Design Framework II product is an open system. You can enter your own design data with industry-standard data formats and use the Design Framework II tools to complete your job.

The design data translators let you translate a design in one format to a Cadence database in Design Framework II format or to take a Cadence database and translate it into another format. Cadence offers a variety of translators to use in integrating your work with the Design Framework II format.

Understanding the Translators

The design data translators convert designs from Design Framework II format to another format, and from another format to Design Framework II format.

These translators fall into three categories: general, netlist, and physical translators. The general translators translate EDIF and Verilog files to and from Design Framework II format. The netlist translators translate netlist designs to and from Design Framework II format. The physical translators translate physical (layout) designs to and from Design Framework II format.
General Translators

The general translators include:

- EDIF 200 Reader, which translates files from EDIF 200 format to Design Framework II database format. For information about EDIF 200 Reader, see the *Virtuoso EDIF 200 Reader and Writer User Guide*.

- EDIF 200 Writer, which translates files from Design Framework II format to EDIF 200 format. For information about EDIF 200 Writer, see the *Virtuoso EDIF 200 Reader and Writer User Guide*.

- PR Flatten, which translates an EDIF or Cadence schematic to a Design Framework II autoLayout database. For information about PR Flatten, see the *Virtuoso Preview Fundamentals Reference*.

- Verilog In, which imports a Verilog HDL description to a Design Framework II database. For information about Verilog In, see the *Verilog In for Design Framework II User Guide and Reference*.

Netlist Translators

The netlist translators include:

- Virtuoso® CDL In, which translates CDL netlists to Design Framework II CDB database format. For information about CDL In, see Chapter 5, “CDL Translators”.

- Virtuoso® CDL Out, which translates a Design Framework II schematic design to a hierarchical netlist in Circuit Definition Language (CDL) format used in Dracula. For information about CDL Out, see Chapter 5, “CDL Translators”.

Physical Translators

The physical translators include:

- Virtuoso® Stream In, which translates Graphics Design Station II (GDSII) Stream mask data from Stream format to Design Framework II database format. For information about Stream In, see Chapter 3, “Translating Stream Files”.

- Virtuoso® Stream Out, which translates files from Design Framework II database format to Graphics Design Station II (GDSII) Stream mask data format. For information about Stream Out, see Chapter 3, “Translating Stream Files”.

Design Data Translator's Reference
Overview of the Translators

- **DEF In**, which reads a Design Exchange Format (DEF) format design description to the dfII database. For information about DEF In, see Chapter 7, “Translating LEF and DEF Files”.

- **DEF Out**, which creates a Design Exchange Format (DEF) format design file from the dfII design. For information about DEF Out, see Chapter 7, “Translating LEF and DEF Files”.

- **LEF In**, which reads a Library Exchange Format (LEF) format library description file and stores the information in a dfII library. For information about LEF In, see Chapter 7, “Translating LEF and DEF Files”.

- **LEF Out**, which creates a Library Exchange Format (LEF) format library description file from a dfII library. For information about LEF Out, see Chapter 7, “Translating LEF and DEF Files”.

- **Virtuoso® CIF In**, which translates Caltech Intermediate Format (CIF) mask data from CIF format to Design Framework II database format. For information about CIF In, see Chapter 4, “Translating CIF Files”.

- **Virtuoso® CIF Out**, which translates files from Design Framework II database format to Caltech Intermediate Format (CIF) mask data format. For information about CIF Out, see Chapter 4, “Translating CIF Files”.

### How to Start the Translators

You can run all the design data translators in interactive mode by using the translators’ user interface forms. You can also run most of the translators in batch mode by entering a command on the command line. See the chapter on a particular translator to see if you can use a command. You access the translator forms from a menu in the Command Interpreter Window (CIW).
The CIW, shown below, displays a log of events when you run any of the Cadence applications, including one of the design data translators.

When you select File from the CIW banner, a pull-down menu appears. From this menu you can select Import or Export. The following diagram shows the Import translators.
If you select *Export* from the *File* pull-down menu, the export translators appear on the pull-down menu.
Preparing to Use Stream and CIF Translators

This chapter describes the following:

- **Overview** on page 24
- **Scaling Your Design** on page 24
- **Using Map Files When Importing** on page 26
- **Using Map Files when Exporting** on page 36
- **Avoiding Translation Errors for Parameterized Cells** on page 45
Overview

Before you run these translators, you might want to make some changes or set some parameters to enhance your results:

- To change the grid resolution (scale) of your library or the Design Framework II library, see “Scaling Your Design” on page 24.
- To create a file to map certain objects in your input library to objects in the Design Framework II library, see “Using Map Files When Importing” on page 26.
- To create a file to map certain objects in the Design Framework II library to objects in your output library, see “Using Map Files when Exporting” on page 36.
- To create parameterized cells (pcells) for Stream, see “Avoiding Translation Errors for Parameterized Cells” on page 45.

Scaling Your Design

If you want your design to be at a different scale or grid resolution in the format you are translating to, you can adjust the ratio of database units to user units. This section describes how to scale a Design Framework II file. You can scale the design before you translate it to a new format or after you translate it to Design Framework II format.

Scaling Process

To scale a Design Framework II design, you can use the scale program that is in your dfii/tools/bin directory. Follow these steps to scale the library.

1. Copy your library to another directory.
   - You need to preserve your original library because the scale program overwrites the library it scales.

2. In a shell (or an xterm) window, change to the directory where the copied library is and type scale at the command line.
   - The system prompts you for a library name, view name, and the ratio of database units to user units. If you leave View Name blank, the system prompts you for the view type.

3. Type the library name, view name or view type of the cells to be scaled, and the DBUPerUU value you want to use.
   - To calculate the value for DBUPerUU, use the following formula:
scale_factor * DBU/UU = DBUPerUU

For example, to reduce a design with a DBU/UU value of 1000 by 20 percent, the value is calculated as:

\[ 0.80 \times 1000 = 800 \]

Here are some sample values you can use.

<table>
<thead>
<tr>
<th>Percentage to scale</th>
<th>Setting for Scale UU/DBU Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>80%</td>
<td>800</td>
</tr>
<tr>
<td>100%</td>
<td>1000</td>
</tr>
<tr>
<td>120%</td>
<td>1200</td>
</tr>
</tbody>
</table>

4. To adjust the grid resolution to the same unit ratio as the original design, reset the DBU/UU layout view property for the translated library by using the Library Property Editor in the Library Manager. Reset the value to 1000.

**Note:** Scaling does not work on pcells.

**Scaling Example**

For example, use the 4.4 lab database files of the Layout Editor training course to create a layout directory. Go into the layout directory and start whatever your command is to bring up the Design Framework II product.

1. To run the `scale` program, after the UNIX prompt, type `scale`.

   You enter `scale` mode and it asks for the following information:

<table>
<thead>
<tr>
<th>Library name</th>
<th>LE</th>
</tr>
</thead>
<tbody>
<tr>
<td>View name (or view type)</td>
<td>layout</td>
</tr>
<tr>
<td>DBUPerUU for the layout view name</td>
<td>200</td>
</tr>
</tbody>
</table>

   You enter the values for your design.

2. Start the Design Framework II product with your usual command.

   For example,
   
   `icfb &`

3. Open the `2nand` layout cellview in the LE library, and find the width of the blue path.
For example,

2 units

4. Start the Library Manager from the CIW menu.

5. Click the middle mouse button on the library name, and release the mouse button on Properties.

   The Library Property Editor form appears.

6. Move the cursor to the DBUPerUU field.

7. To expand the field into a form, click on the DBUPerUU field.

   Note that maskLayout is 200.

8. Change maskLayout to 1000.

   Note that the layout designs in your library are scaled down to 80 percent.

9. Open your design again.

   Note that the width of the path changed to 0.4 units.

The new library is scaled and ready to translate. You can now copy the library back to the original directory or translate it.

**Using Map Files When Importing**

Map files tell the Stream, CIF translators how to map certain objects and properties to Design Framework II objects and properties.

The form and organization of map files are as follows:

- Map files are ASCII tables you can create with any text editor.
- A number field can contain one number, two or more numbers separated by commas (a series), pairs of numbers separated by hyphens (ranges), or combinations of series and ranges.
- Make each mapping entry a single line that is no more than 256 characters.
- Each mapping entry in cell map file can be up to 1024 characters long.
- Separate fields with one or more spaces or tabs.
- Place comments on separate lines beginning with the pound sign (#), the semicolon (;), or the exclamation point (!) as the first character of the line.
Insert blank lines anywhere; the translator ignores them.

The following table shows the map files that are available for each translator:

<table>
<thead>
<tr>
<th>Mapping File</th>
<th>Stream</th>
<th>CIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell name mapping file</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Layer name mapping file</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Text font mapping file</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>User-defined property mapping file</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

**Cell Name Mapping File**

A cell name mapping file contains a table that maps Stream, CIF cell names to Design Framework II database cellview names.

You can also specify cell name mapping information in a SKILL file. You specify the name of the SKILL file in the *User-Defined Data* form for the translator. If you do not provide cell mapping information, the translator uses the *Case Sensitivity* option setting from the *User-Defined Data* form. For each cell name, the translator follows the procedure shown in the *Mapping Cell Names* figure on page 28.
The following is an example of an *Import – Stream* cell name map file.

<table>
<thead>
<tr>
<th>#Cadence Cell</th>
<th>Cadence View</th>
<th>Translated Cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>alu</td>
<td>layout</td>
<td>ALU_CHIP</td>
</tr>
<tr>
<td>adder</td>
<td>layout</td>
<td>ADDER</td>
</tr>
</tbody>
</table>

**Layer Name Mapping File**

You can provide the information about mapping Design Framework II layer-purpose pairs with Stream and CIF layers in the following files:

- Layer Name Mapping File
- User-Defined SKILL File
- Technology File
If the Stream translation rules are not specified in any of these files, PIPO uses its internal layer mapping mechanism.

**Layer Name Mapping File**

The layer name mapping file contains a table that maps Stream, CIF, and layer and data type numbers to Design Framework II layer-purpose pairs.

**Stream Layer Name Mapping File**

A layer mapping file contains four entries: the layer name and layer-purpose of the Design Framework II database layer name, a Stream layer number, and a Stream data type. The layer map file needs to follow these basic rules:

- Stream layer numbers must be integers between 0 and 255, inclusive.
- Data type numbers must be integers between 0 and 255, inclusive.
- You can map more than one Stream layer to one Design Framework II layer-purpose pair.

The following is an example of an *Import – Stream* layer map file.

<table>
<thead>
<tr>
<th>#Cadence Layer Name</th>
<th>Cadence Layer Purpose</th>
<th>Stream Layer Number</th>
<th>Stream DataType</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>metal</td>
<td>drawing</td>
<td>13,10,18</td>
<td>0–63</td>
</tr>
<tr>
<td>poly</td>
<td>boundary</td>
<td>40</td>
<td>1</td>
</tr>
</tbody>
</table>

**Note:** PIPO can translate any of the mapped Stream layer-datatype pairs to any valid Design Framework II layer-purpose pairs.

**CIF Layer Name Mapping File**

If the CIF layer name is not a number, but a name such as poly or metal, *Import – CIF* uses the layer name as the Design Framework II layer name. If the layer is not in the output Design Framework II library, *Import – CIF* creates a new layer name and chooses a layer number based on the sequence of the layers in the input file.

You can map more than one CIF layer to a Design Framework II layer-purpose pair.
The following is an example of an *Import – CIF* layer mapping file:

<table>
<thead>
<tr>
<th>#Cadence Layer Name</th>
<th>Cadence Layer Purpose</th>
<th>CIF Layer Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td></td>
<td></td>
</tr>
<tr>
<td>substrate</td>
<td>drawing</td>
<td>POLY</td>
</tr>
<tr>
<td>ntub</td>
<td>boundary</td>
<td>METAL, LL</td>
</tr>
<tr>
<td>thinox</td>
<td>cell</td>
<td>5, L4, L5</td>
</tr>
</tbody>
</table>

**User-Defined SKILL File**

You can also put the layer name mapping information in a SKILL file. However, if you specify a layer name mapping file, the translators ignore the SKILL file for layer mapping.

You specify the name of the SKILL file in the User-Defined Data form for the translator.

**Technology File**

If you do not provide mapping information in either a layer name mapping or a SKILL file, and the technology library contains a Stream translation rules section (*streamLayers*), *Import – Stream* translates only those Stream layers and Stream datatypes for which you set the `translate` property to `t`. If you set the `translate` property to `nil`, Stream layer and Stream datatype are dropped and a warning message is displayed.

**Note:** The *streamLayers* subclass occurs in the Layer Rules class in the technology file.

A sample of the *streamLayers* section from the technology file is shown below.

```plaintext
streamLayers(
  ; ( layer streamNum dataType translate )
  ( ndiff 1 0 t )
  ( pdiff 2 0 t )
  ( pwell 6 0 t )
  ( metall 45 0 t )
)
```

For more information about setting Stream translation rules, see the section "Defining Stream Translation Rules" in *Chapter 4 of the Technology File and Display Resource File User Guide*. 
Internal Layer Mapping Mechanism

If you do not set Stream translation rules in the technology file and do not provide mapping information in either a layer map file or a SKILL file, then PIPO uses its internal layer mapping mechanism. Stream layers and Stream datatypes are automatically mapped as:

L#StreamLayerNo. P#StreamDatatypeNo.

For example, if the input layer number is 20 and datatype is 15, then after mapping, dfII layer name is L20 and purpose name is P15.

New layers created by PIPO have the same dfII layer numbers as the Stream layer numbers. If PIPO cannot assign newly created layer the same dfII layer number as the Stream layer number, then PIPO internally assigns a free dfII layer number (for example, Stream layer numbers 240 to 250). Similarly, PIPO assigns same dfII purpose number for the Stream datatype.

DfII supports layer numbers 0 - 255. If the Stream layer number is 264, then the Cadence layer number L264 is created on a free Cadence layer number.

If there are more than 255 distinct layer numbers in the Stream file, then the automatic layer mapping of Stream In is able to translate only 255 distinct layers. Data on rest of the Stream layers is dropped. In such cases, if you want to translate the entire data, then provide a layer mapping file with multiple Stream layers mapped to single Cadence layer.

Note:

- PIPO always maps Stream datatype number 0 to the purpose drawing. All other datatypes are mapped according to the datatype numbers.
- All Stream datatypes are mapped to Cadence purpose drawing if the mergeUndefPurposToDrawing option is selected.
- Consider the layer map entry (metal1 drawing 4 0). If the target technology library does not contain layer metal1, then this layer will be created by PIPO if the technology library is writable on Cadence user layer number 4. If Cadence user layer number 4 is not free, then metal1 is created on any other free Cadence layer number.
- PIPO Stream In sets visibility, selectability, priority, and display packet attributes for the LPPs that are created by PIPO.

Same Stream Number and Data Type Pair Mapped to Multiple Layer-Purpose Pairs

In either the layer name map or technology file, the same Stream layer number and data type pair might be mapped to more than one Design Framework II layer-purpose pairs. In this
scenario, Stream In translates all the entries for that Stream layer only to the first Design Framework II layer-purpose pair specified in the layer map or technology file.

If you want Stream In to retain the different layer-purpose pairs mapped to the same Stream layer number and data type pair, update the layer name map file used during the Stream Out stage itself. Change either the Stream layer number or the Stream data type for all such layer-purpose pairs. Use the updated layer name file to again run Stream Out and then Stream In to obtain the desired results.

For example, if the layer map file looks like:

<table>
<thead>
<tr>
<th>Cadence Layer Name</th>
<th>Cadence layer-purpose</th>
<th>Stream Layer Number</th>
<th>Stream DataType</th>
</tr>
</thead>
<tbody>
<tr>
<td>metal1</td>
<td>drawing</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>metal2</td>
<td>boundary</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>metal3</td>
<td>boundary</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

where, metal1 and metal2 have the same Stream layer number and data type. Even though metal3 has the same Stream layer number but it differs from metal1 and metal2 in the Stream data type. So, Stream In translates the Stream layer number 2 with data type 0 only to the first layer-purpose pair mapped to this Stream layer and data type, that is, metal1 drawing. Stream In translates Stream layer number 2 with data type 1 to metal2 boundary.

### Text Font Mapping File for Stream In

The text font mapping file contains a table that maps text fonts to Design Framework II database text fonts.

You can also put text font mapping information in a SKILL file. However, if you specify a text font mapping file, the translator ignores the SKILL file for text font mapping. You specify the name of the SKILL file in the User-Defined Data form for the translator.

The Design Framework II font must be one of the following:

- eurostyle or europe
- script
- raster
- math
Mapping Fonts with Stream

When importing Stream files, use the following equation to map fonts:

\[ \text{Font\_Height} = \text{MAG} \times \text{scale\_factor} \]

The variables in the equation can be defined as follows:

- **MAG**: Cannot store more than 6 places after the decimal point. That is, any places after \(x.xxxxxx\) are truncated.
- **Font\_Height**: Cannot store more than 3 places after the decimal point. That is, any places after \(x.xxx\) are truncated.

When exporting Stream files, use the following equation to map fonts:

\[ \text{MAG} = \frac{\text{Font\_Height}}{\text{scale\_factor}} \]

The default scale factor is 1.

**Note**: Valid Stream font numbers are 0, 1, 2, and 3. If you use any other number, the resulting Stream font number is unpredictable. For more information about Stream font numbers, see Appendix A Stream Format.

The following example maps four Cadence fonts to Stream fonts:

<table>
<thead>
<tr>
<th>#Cadence Font</th>
<th>Stream Font</th>
<th>Scale Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>stick</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>script</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>roman</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
During Stream In, if a font is not mapped using a text font map file, then following default mapping is used for this font:

<table>
<thead>
<tr>
<th>dfII Font</th>
<th>Stream Font</th>
</tr>
</thead>
<tbody>
<tr>
<td>stick</td>
<td>0</td>
</tr>
<tr>
<td>euro</td>
<td>1</td>
</tr>
<tr>
<td>gothic</td>
<td>2</td>
</tr>
<tr>
<td>math</td>
<td>3</td>
</tr>
</tbody>
</table>

**Text Font Mapping File for CIF**

The following equation applies to CIF translations:

\[ \text{label\_size} = \text{MAG} \times \text{font\_height} \times \text{scale\_factor} + 0.5 \]

The variables in the equation can be defined as follows:

- **MAG**: Read from the TEXT record in the CIF database
- **font\_height**: The value in the text font map file
- **scale\_factor**: Calculated as follows:

\[ \text{scale\_factor} = \text{unit\_factor} \times \frac{\text{Cadence\_scale}}{\text{trans\_scale}} \]

The variables in the equation can be defined as follows:

- **unit\_factor**: The factor of unit change (for example, micron to mil)
- **Cadence\_scale** and **trans\_scale**: The number of user units per database unit in the respective files

The following is an example of a text font mapping file:

<table>
<thead>
<tr>
<th>#Cadence Font</th>
<th>Original Font</th>
<th>Font Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stick</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>script</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Property Mapping File for Stream In

The optional property mapping file contains a table to map object types and property names from the Stream file to the Design Framework II database. If you do not specify a file name, Import – Stream tries to locate property name and value pairs in the Stream file by searching for equal signs (=) and a character separator, usually a comma (,). You can specify the character Import – Stream uses to separate the property string with the User-Defined Property Separator option.

A property mapping file must contain entries with the following syntax:

```
attNum objectType propertyName
```

The variables in the equation can be defined as follows:

- **attNum**: Attribute number of the input property. Start your entry with this value in the first column of the line.
- **objectType**: Type of object.
  - Valid Values: rect, polygon, dot, inst, mosaic, path, pin, line, label, donut, ellipse, arc
- **propertyName**: The name of the property to create for the attribute number of the object type.
  - Valid Values: any valid ASCII string

The following is an example of a property mapping file:

```
# Att Num Object Type Property Name
# Cadence Font Original Font Font Height
roman 2 1

#Att Num Object Type Property Name
12 pin "STREAM Property 12"
125 pin NEW_PROPERTY_NAME
124 rect "Any Valid Name"
```
If you do not define an attribute number or object type in the map file, Import – Stream uses the default property name "STREAM PROPERTY #" where # is the attribute number.

Caution

If the input STREAM file contains attributes or properties in the format name=value (generated using the -f p option of StreamOut), then the user should not use the property map table with StreamIn because it automatically searches for equal signs (=) and extracts property names and values distinctly. Using a property map table in this case, results in the property values being directly translated as input attributes themselves. For Example, width = "width=5".

Using Map Files when Exporting

Map files tell Stream, CIF how to map certain Design Framework II objects and properties to the translated format for objects and properties.

The form and organization of map files is as follows:

- Map files are ASCII files you can create with any text editor.
- A number field can contain one number, two or more numbers separated by commas (series), pairs of numbers separated by hyphens (ranges), or combinations of series and ranges.
- Make each mapping entry a single line that is not more than 256 characters.
- Separate fields with one or more spaces or tabs.
- Place comments on separate lines beginning with the pound sign (#), semicolon (;), or exclamation point (!).
- Insert blank lines anywhere; the translators ignore them.

The following table shows the map files that are available for each translator:

<table>
<thead>
<tr>
<th>Mapping File</th>
<th>Stream</th>
<th>CIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell name mapping file</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Layer name mapping file</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Text font mapping file</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>User-defined property mapping file</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
Cell Name Mapping File for Stream Out

A cell name mapping file contains a table that maps the Design Framework II database cellview names to specified cell names.

You can also specify cell name mapping information in a SKILL file. You specify the name of the SKILL file in the User-Defined Data form for the translator. If you do not provide cell mapping information for a cell in either a cell mapping file or a SKILL file, the translator uses the Case Sensitivity option setting from the User-Defined Data form. For a flow tree of how the translators determine how to translate a cell name, see the Mapping Cell Names figure on page 28.

The following is an example of a cell map file:

<table>
<thead>
<tr>
<th>#Cadence Library</th>
<th>Cadence Cell</th>
<th>Cadence View</th>
<th>Translated Cell</th>
</tr>
</thead>
<tbody>
<tr>
<td># Name</td>
<td>Name</td>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td># LIB_1</td>
<td>ALU_CHIP</td>
<td>layout</td>
<td>alu</td>
</tr>
<tr>
<td>refLib</td>
<td>ADDER</td>
<td>layout</td>
<td>adder</td>
</tr>
</tbody>
</table>

Layer Name Mapping File for Stream Out

The layer name map file maps Design Framework II layer-purpose pairs to the specified values.

You can also specify layer name mapping information in a SKILL file; however, if you specify a layer name map file, the translators ignore the SKILL file for layer name mapping. You specify the SKILL file in the User-Defined Data form for the translator.

Mapping Layers with Stream

If you do not provide mapping information in either a layer name map file or a SKILL file, Export – Stream translates only the layers for which you set the translate property to t in the Stream translation rules in the technology file. If the translate layer rule is set to nil,
or no layer rule corresponding to a layer is defined, the translator drops that layer. For more information about setting Stream translation rules, see the section "Setting Stream Translation Rules for a Layer" in the Technology File and Display Resource File User Guide.

Each line in the layer name map file contains four entries: a Design Framework II database layer name, a Design Framework II database layer-purpose, a Stream layer number, and a Stream data type.

The Stream layer name map file follows these basic guidelines:

- Stream layer numbers must be integers between 0 and 255, inclusive. Layers 195 to 255 are reserved system layers in Virtuoso.
- Data type numbers must be integers between 0 and 255, inclusive.
- You can use any Design Framework II layer-purpose name.
- You can map a Design Framework II layer-purpose to more than one Stream layer. Separate the ranges with hyphens and the individual numbers and ranges with commas.
- The layer map file must contain all layers that occur in the data translation. Export – Stream converts only layers listed in the layer map file. (It does not convert layers that are not listed in the layer map file.)

The following is an example of an Export – Stream layer mapping file:

<table>
<thead>
<tr>
<th>DFII</th>
<th>DFII Layer Name</th>
<th>Stream Layer Number</th>
<th>Stream Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATL1</td>
<td>drawing</td>
<td>1,2</td>
<td>0</td>
</tr>
<tr>
<td>MATL1</td>
<td>drawing</td>
<td>3,4-9</td>
<td>0</td>
</tr>
<tr>
<td>text,diff</td>
<td>drawing</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>metal,pwell</td>
<td>drawing</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

You can map a dfII layer-purpose to more than one stream layer by also using the following format in the stream translation rules in the technology file associated with the cellview/library:

```plaintext
;( layer "<layerName>" "<purpose>" (<num1> <num2>...) (<num1> <num2>) t)
```

July 2007 38 Product Version 5.1.41
The actual mapping is explained with the following cases:

Case I:

If the streamLayers rule in the techfile is:

```
; ( layer streamNumber dataType translate )
( ("poly" "drawing") 20 l t )
```

StreamOut will transfer all the data on layerPurpose ("poly" "drawing") into one layer in output Stream file as

(a) Stream Layer No: 20, Stream Datatype: 1

Case II:

If the streamLayers rule in the techfile is:

```
; ( layer streamNumber dataType translate )
( ("poly" "drawing") 20 (20 30) 1 t )
```

StreamOut will transfer all the data on layerPurpose ("poly" "drawing") into two duplicates in output Stream file as

(a) Stream Layer No: 20, Stream Datatype: 1
(b) Stream Layer No: 30, Stream Datatype: 1

Case III:

If the streamLayers rule in the techfile is:

```
; ( layer streamNumber dataType translate )
( ("poly" "drawing") 20 (1 2) t )
```

StreamOut will transfer all the data on layerPurpose ("poly" "drawing") into two duplicates in output Stream file as

(a) Stream Layer No: 20, Stream Datatype: 1
(b) Stream Layer No: 20, Stream Datatype: 2

Case IV:

If the streamLayers rule in the techfile is:

```
; ( layer streamNumber dataType translate )
( ("poly" "drawing") 20 (20 30) (1 2) t )
```

StreamOut will transfer all the data on layerPurpose ("poly" "drawing") into four duplicates in output Stream file as
(a) Stream Layer No: 20, Stream Datatype: 1
(b) Stream Layer No: 20, Stream Datatype: 2
(c) Stream Layer No: 30, Stream Datatype: 1
(d) Stream Layer No: 30, Stream Datatype: 2

Note: Design Framework II Import – Stream can translate layer numbers 0 to 255 and data types from 0 to 255.

During Stream Out, the explicitly stated layer-purpose pair overrides the other entries for that layer. For example, consider the following entries:

```
streamLayers(
  ;( layer       streamNumber    dataType        translate )
  ;( -----       ------------    --------        --------- )
  (IN1        1 1 nil )
  ("IN1" "P1") 0 1 t )
)
```

In the above case, PIPO would respect the second entry and ignore the first entry because the second entry explicitly states the IN1P1 layer-purpose pair be translated to a stream file. The first entry, being implicit, is ignored by PIPO.

Consider another example.

```
streamLayers(
  ;( layer       streamNumber    dataType        translate )
  ;( -----       ------------    --------        --------- )
  (IN1        0 1 nil )
  ("IN1" "P1") 1 1 t )
)
```

In this case, PIPO would respect the first entry because first entry explicitly states the IN1P1 layer-purpose pair be translated to a stream file. The second entry, being implicit, is ignored by PIPO. Therefore, the order in which the entries are specified does not impact the translation.

However, for conflicting entries PIPO translates the first entry in the file. For example, consider the following entries:

```
streamLayers(
  ;( layer       streamNumber    dataType        translate )
  ;( -----       ------------    --------        --------- )
  ("IN" "P1") 0 0 nil)
  ("IN" "P1") 0 0 t)
)
```
In this case, PIPO would switch off the translation process for the IN P1 layer-purpose pair because in the case of conflicting entries only the first entry is executed. PIPO ignores the subsequent entries.

```
streamLayers(
  ( layer streamNumber dataType translate )
  ( IN 1 0 t )
  ( IN 0 0 nil)
)
```

Similarly, in the above example, PIPO would translate the IN P1 layer-purpose pair and ignore the second entry.

**Mapping Layers with CIF**

If you do not provide mapping information in either a layer map file or a SKILL file, *Export – CIF* creates layers based on the order in which the Design Framework II layers are encountered. *Export – CIF* uses the Design Framework II layer name as the CIF output layer name.

Here is an example of an *Export – CIF* layer mapping file.

<table>
<thead>
<tr>
<th>#Cadence Layer Name</th>
<th>Cadence layer-purpose</th>
<th>CIF Layer Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ntub,diff</td>
<td>drawing</td>
<td>METAL</td>
</tr>
<tr>
<td>text</td>
<td>label</td>
<td>DIFF</td>
</tr>
</tbody>
</table>

**Text Font Mapping File for Stream Out**

The text font mapping file maps dfII database text font names to Stream text font numbers. Each line contains three entries: a dfII font name, a Stream font number, and a scale factor.

You can also specify text font name mapping information in a SKILL file. However, if you specify a text font name file, the translator ignores the SKILL file for text font name mapping. You specify the SKILL file in the *Export – Stream User-Defined Data* form.

The dfII text font name must be one of the following:

- eurostyle
- europe
- raster
Note: Legal Stream font numbers are 0, 1, 2, and 3. If you use any other number, the resulting Stream font number is unpredictable. For more information about Stream font numbers, see Appendix A Stream Format.

During Stream Out, if a font is not mapped using a text font map file, then following default mapping is used for this font:

<table>
<thead>
<tr>
<th>dfII Font</th>
<th>Stream Font</th>
</tr>
</thead>
<tbody>
<tr>
<td>stick</td>
<td>0</td>
</tr>
<tr>
<td>euro</td>
<td>1</td>
</tr>
<tr>
<td>gothic</td>
<td>2</td>
</tr>
<tr>
<td>math</td>
<td>3</td>
</tr>
</tbody>
</table>

The following is an example of an Export – Stream text font name mapping file:

<table>
<thead>
<tr>
<th>dfII Font Name</th>
<th>Stream Font Number</th>
<th>Scale Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>stick</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>script</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>roman</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Property Mapping File for Stream Out

A property name mapping file is an optional file that you can use to map object types and property names from the Design Framework II database to the Stream file. You can specify the character Export – Stream uses to separate the property string with the User-Defined Property Separator option.
Cellview level properties are lost in StreamOut because the stream format does not support structure level attributes.

The property name mapping file must contain entries using the following syntax:

-options
 attNum objectType propertyName, ...

-options Specify the format of the output and the character separator to use between property values. You must start your option entry with a - (hyphen) in the first column of the line.

Valid Values:

- f v Format the output to show the property values only. This shows "inputOutput" in the Stream file.

- f p (default setting) Format the output to show the property’s name-value pair. This shows "accessDir=inputOutput" in the Stream file.

- s ‘c’ (or "c") (default is a comma: ‘,’) Separate the property values or property-value pairs in the output with the character indicated by c. You can also use double-quotes (""") to indicate c. The default separator is a comma (,).

attNum Attribute number you assign to the property. Start your entry with this value in the first column of the line.

objectType Type of object.
Valid Values: rect, polygon, dot, inst, mosaic, path, pin, line, label, donut, ellipse, arc

propertyName Names of the object-type properties and attributes,
Valid Values: any valid property name and the following attributes:
For shapes: objTypes, LyrName
For instances: objTypes, instName
For pins: objTypes, pinName, accessDir, termName, direction, instName
Preservation of instance names in PIPO

In PIPO, `instName` is treated as a special property name. StreamOut writes the instance name as the value of property `instName` if specified by the user in the property map table. StreamIn restores the same instance names in the output database by reading these attributes from stream file. In the absence of the `instName` property, the instance names are not preserved in PIPO.

The following is an example of a property name mapping file:

```
-f v -s ' '
12    pin    pinName,accessDir
125   pin    termName,direction
124   rect   objTypes,LyrName
125   rect   sigName
```

Caution

*If the same attribute number is associated with different user property names of the same object, then the properties will get concatenated in the output stream file. In this case, you may not be able to stream-in GDSII data back to CDBA correctly using the property map table. However, you can have the same user property name associated with two different attribute numbers as long as the object type is different.*

Pin Text Mapping File for Stream Out

A pin text map file is an optional file you use when you want to convert pins to labels. The pin text file maps the parameterized cell types to the layer on which you want the label.

If your design contains geometry pins, you do not need to create entries for them in the pin text map file. The text for a geometry pin is mapped to the same Stream number and data type as other objects of the same shape and layer.

The following is an example of a pin text mapping file:

```
#Pin Type    Layer Number
#
```
The layer numbers that you provide to map the symbolic pins are database layer numbers. The symbolic pins finally appear as text on the Stream layer mapped to the database layers that you specify in the pin text map file.

If the pin is on a net, the translator uses the net name for the label. Otherwise, the translator uses the pin name.

### Avoiding Translation Errors for Parameterized Cells

To avoid translation errors such as "undefined function" for designs containing parameterized cells (pcells), you need to create pcells in accordance with the safety rules defined in the Parameterized Cell Functions chapter of the Custom Layout SKILL Functions Reference Manual. If your pcells use SKILL functions that are unsupported or not intended for use in pcells, or both, the pcell code might fail when you try to translate your design to a format other than the Design Framework II format.
Translating Stream Files

This chapter describes the following:

- Overview on page 48
- Translating Reference Libraries Using Stream on page 48
- Retaining Reference Library Structure in Stream Files on page 49
- Moving Stream Files between Tape and Disk on page 52
- Using Import – Stream on page 53
- How Import – Stream Translates Data on page 82
- Import – Stream Output Files on page 84
- Using Export – Stream on page 91
- How Export – Stream Translates Data on page 127
- Export – Stream Output Files on page 130
- Performance Guidelines for PIPO Stream In and PIPO Stream Out on page 139
- Running 64 bit PIPO - Stream In/Stream Out on page 145
- Streaming Out Large Designs Using 32-Bit PIPO on page 146
- Keep PCell Permutations in Stream Out/Stream In on page 148
Overview


For information about using the Stream translators in the context of laying out a design, refer to Virtuoso Layout Editor User Guide.

Translating Reference Libraries Using Stream

When you translate a Design Framework II library, you create a single Stream file. If your design includes cells from reference libraries, you can choose to merge the structure definitions of all the cells in the design into the Stream file. Otherwise, you can translate the structure definition of a design cell and only the cell name and placement information for the cells defined in reference libraries.

This section provides information about how to

- Retain reference library structure in your Stream files
- Translate a Stream file that does not contain referenced cells to a Design Framework II library that uses reference libraries
- Translate a Stream file that contains referenced cells to the Design Framework II format

You specify how you want to translate referenced cells by setting the Retain Reference Library option on the Stream In and Stream Out Options forms. For example, if you want to translate abstracts so that you can run a place-and-route application, you probably do not need detailed cell definitions. To do this, turn the Retain Reference Library option on. This speeds the translation process and reduces the size of your Stream file. However, if you want to translate a design, you can run the Layout Versus Schematic (LVS) tool, you need complete definitions for all the cells in your design and must turn the Retain Reference Library option off.

For detailed information about the Virtuoso® Stream In form, see the section “Stream In Form” on page 57. For detailed information about the Virtuoso® Stream Out form, see the section “Stream Out Form” on page 95.

If you want to retain reference library structure in your Stream and Design Framework II files, keep the following in mind:

- The library path you define in the cds.lib file is used to search the cell.
Important

The cds.lib file must contain a minimal list of libraries for Stream In to perform faster. The minimum number of libraries listed in the cds.lib file must be the target library and the libraries whose cells are instantiated in the stream file.

Before you translate designs containing reference library cells, make sure your cells have unique names, even if they are in different libraries. If you translate a Design Framework II database file that contains two cells from different libraries but with the same cell name, Export – Stream changes the names in the Stream file. When you use the file in another application, or when you translate it back to the dfII format, the application is not able to locate the cells in the reference library because the names have been changed.

Export – Stream writes a warning message to your error file when you translate a cell that uses more than two reference libraries. Export – Stream still translates the cell names and placement information for the cells from the remaining reference libraries.

When you translate a Stream file to dfII format, you need to know whether it contains complete information or reference information. If you created the Stream file so that it retains the reference library structure, the referenced cells are not completely defined and you must translate the file to dfII format with the Retain Reference Library option checked.

Retaining Reference Library Structure in Stream Files

You can choose to maintain the reference library structure in individual Stream files, or you can merge the complete cell definitions from several libraries into a single Stream file. For example, the following mux2 design contains three nand2 cells and an Inv cell that are defined in the master reference library.

Sample Design Framework II Library Structure

Library: tutorial
Cell: mux2

Library: master
Cell: Inv
Cell: nand2
When you translate your dfII library to Stream format, you can generate two different Stream files depending on whether you select the *Retain Reference Library* option in the *Stream Out Options* form.

- If you select the *Retain Reference Library* option, your Stream file contains complete structure definitions for the geometric and text objects in your design, as well as the instances placed from the current library. Instances placed from other libraries are translated with only the following information:
  - Cell name
  - Origin
  - Orientation
  - Magnification
  - Number of rows and columns, if an array value

- If you do not select the *Retain Reference Library* option, your Stream file contains complete structure definitions for all objects in your design. *Export – Stream* merges the structure definitions from your reference libraries into the Stream file to define the referenced cells. Complete structure definitions contain the following information:
  - Cell name
  - Origin
  - Orientation
  - Magnification
  - Number of rows and columns, if an array
  - Creation date
  - Date of last change
  - Object type
  - Stream layer number
  - Stream data type
  - Coordinates for all points of the object
  - Properties set on the object, if any
Caution

Creation date of a cellview is not stored in the dfll database. Therefore, Stream In and Stream Out do not preserve this field. By default, Stream Out sets it to Jan 1, 1970. This field in the GDS file is ignored by Stream In.

Translating a Stream File with Reference Information

If the Stream file retains reference library structure, the file does not contain complete structure definitions for the referenced cells. To translate the file, you need to do the following:

1. If you are creating a new library, list the reference libraries in the cds.lib file.
   This list helps the translated cell locate the referenced cells.

   Important

   The cds.lib file must contain a minimal list of libraries for Stream In to perform faster. The libraries listed in the minimal cds.lib should be the target library and the libraries whose cells are instantiated in the stream file.

2. If the library you are translating to already exists, make sure the reference libraries you want to use are listed in the cds.lib file.
   - From the CIW menu, choose Tools – Library Manager.
   - From the Library Manager menu, choose Edit – Library Path.
   - Check the list of libraries.
   - If necessary, add the reference libraries.
   - In the CdsLibEditor form, choose File – Save.
   - To exit the CdsLibEditor form and the Library Manager, choose File – Exit.

3. In the Options form of Import – Stream, select the Retain Reference Library option.

4. Set the other options that you want in the Virtuoso® Stream In form.

5. To translate the Stream file, click OK.

   Import – Stream translates the Stream file to a Design Framework II file that lists the libraries you referenced.
Translating a Stream File with Complete Structures

If the Stream file contains complete structure definitions for all the objects in the design, you have two choices:

- You can translate the Stream file to a complete Design Framework II library that contains cellviews for all the cells in the design.
- You can translate the Stream file to a library that references existing reference libraries for the structure definitions of the referenced cells.

To create cellviews for all cells in the Stream file, select the *Retain Reference Library* option.

*Import – Stream* places all cellviews in the destination library.

To create cellviews that reference your existing libraries for the structure of the cells in the Stream file, follow the procedure in the section “Translating a Stream File with Reference Information” on page 51.

Moving Stream Files between Tape and Disk

*Import – Stream* and *Export – Stream* assume that the Stream file resides on the local disk. If the Stream file is not there, you need to copy it to the local disk. Because the process you use to move the Stream file to and from the disk is system-dependent to an extent, it can be a difficult step in the data translation process.

Users frequently archive GDSII Stream-formatted data on magnetic tapes. For systems with a 1600-bpi tape drive, you can use the UNIX® *dd* command to read or write a Stream tape. A Stream tape must have a 2048-byte block size.

You can also use the UNIX *dd* command to move Stream files between UNIX systems that have floppy disk drives. For example, use the following command to move a Stream file from tape to disk:

```
dd if=/dev/mnt0 of=file.gds bs=2048
```

Use this command to move a Stream file from disk to tape:

```
dd if=file.gds of=/dev/mnt0 bs=2048
```

You can move a Stream file between systems through a local network. If your source and destination machines both use the UNIX operating system, you can use the *rcp* command to transfer the file. If your source and destination systems are different, you need to know about local network capabilities. For information about transferring binary files between different operating systems, refer to your system documentation.
Recreating Relative Object Design Data during Translation Using Non-Cadence Tools

Relative Object Design (ROD) data that is streamed out and later read by a non-Cadence tool can be modified by the non-Cadence tool or by its environment. For example, a non-Cadence tool or its environment might modify a ROD object by moving it or changing its point array, layer, or connectivity information.

After streaming out your data, modifying it using a non-Cadence tool, and then streaming it back in to dfII, the ROD data is restored, and all external editing to the geometric part of ROD data is reflected in the dfII environment, with the following exceptions:

- If a change is made to any subpart of a ROD multipart path, that change is undone. The subparts of multipart paths are created from master path data. Therefore, edits to subparts are ignored by Stream In.

- ROD alignments are always recreated in the original order in which they were created. For example, objectB is aligned to objectA, and then objectA is moved during an external edit. While streaming the data in, the system moves objectB to maintain its alignment with objectA. However, if objectB, instead, was moved during an external edit, streaming in the data would result in the realignment of objectB to objectA.

Note: In a non-Cadence editing environment, you can eliminate the two exceptions listed above by deleting the ROD object name data stored in Property 126, and streaming it back in with the Cadence Stream reader. Referring to the example in the second bullet above, if you do not want to maintain the alignment of objectB to objectA, you can delete Property 126 for either objectA or objectB. This action effectively unnames the ROD object, making it impossible for the system to reestablish ROD alignments for the unnamed object.

For more information on the preservation of ROD data, see the Rod Directory option of the Stream In Options form.

For more information on ROD data, see Virtuoso Relative Object Design User Guide.

Using Import – Stream

You can use Import – Stream to translate GDSII Stream-formatted mask data to Design Framework II database format. In addition, you can use Import – Stream to check geometric data, remap layers, filter Stream files, and create a text version of Stream data.

This section gives an overview of the files Import – Stream uses and produces during translation. Later sections give detailed descriptions of the Import – Stream command and the input and output files.
The following figure shows the files Import – Stream uses and the files it produces during translation.

The following list provides generic information about the files you use for Import – Stream:

- **Stream file**
  This is the Stream-formatted file that you want to translate to a dfII database.

- **Template file**
  This is a file that contains the parameter settings you want to use to run Import – Stream. You can load the parameter settings in the template file into the Virtuoso® Stream In form. If you use the pipo command to translate your Stream file, you must specify a template file. (The pipo command is the noninteractive, batch mode version of Import – Stream and Export – Stream.) You can create a template file by setting the values you want in the Virtuoso® Stream In form and clicking the Save button in the form. Or you can copy the sample template file from samples/transUI/streamin.il and modify it.

- **Technology file**
  This is the technology file you want Import – Stream to use to compile the Design Framework II database. If you do not specify a technology file, Import – Stream uses the default technology file in samples/techfile/default.tf.
Cell name mapping file

This is an optional file you can use to map the Stream cell names to the cell and view names you want to use in the Design Framework II database. For detailed information about creating a cell name mapping file, see the section “Cell Name Mapping File” on page 27.

Layer name mapping file

This optional file maps the layer numbers used in the Stream file to the layer names you want to use in the Design Framework II database. For information about creating a layer name map file, see the section “Layer Name Mapping File” on page 28.

Text font mapping file

This optional file maps the text font numbers used in the Stream file to the text font names you want to use in the Design Framework II database. For information about creating a text font map file, see the section “Text Font Mapping File for Stream In” on page 32.

Property mapping file

This optional file maps object types and property names from the Stream file to the Design Framework II database. For detailed information about creating a property mapping file, see the section “Property Mapping File for Stream In” on page 35.

SKILL file

This optional file maps cell names, text font types, and layer names, and handles illegal shapes, such as open or reentrant polygons. For detailed information about creating a SKILL file, see the chapter “Using SKILL to Customize Stream and CIF Files” on page 269.

Note: Import – Stream reads the SKILL file for a particular type of mapping under certain conditions. For detailed information about when Import – Stream uses the SKILL file, see the section “Using Map Files When Importing” on page 26.

Import – Stream produces an error file and one of three output files: a Design Framework II database, a technology file, or a text interpretation of the Stream file.

Design Framework II database

This is the output Design Framework II database. Import – Stream writes informational, warning, and error messages in the error file during translation, so you can check the error file for problems encountered during, if any, translation.

Technology file

This is a technology file that you can generate and customize before you translate your Stream file. When you use Import – Stream to create an ASCII technology file, Import
- *Stream* does not translate any data. Generating and customizing a technology file before you translate your data is one way to fine-tune how you want your translated library to look. (Another way to do this is to translate the Stream file, generate a technology file from the translated Design Framework II library, customize it, and reload it into your library.)

- **ASCII dump**
  This is a text interpretation of how *Import – Stream* translates the devices in your Stream file. When you produce an ASCII dump, *Import – Stream* does not translate any data. Generating an ASCII dump file is helpful when you are not sure what the Stream file contains.

- **Error file**
  This is the file that contains all the informational, warning, and error messages that *Import – Stream* generates during translation. You can set several parameters on the *Stream In Options* form to filter the type of messages that *Import – Stream* writes to the error file. See *Stream In Options Form*. 
## Import – Stream Forms

Translates files from GDSII Stream format to dfII database (Opus DB).

This section describes the following Import - Stream forms:

- Stream In Form on page 57
- Stream In User-Defined Data Form on page 63
- Stream In Options Form on page 66

For a list of equivalent template file options, refer to the Stream In GUI and Template File Options section.

### Stream In Form

![Stream In Form](image)

- **Template File**
- **Run Directory**
- **Input File**
- **Top Cell Name**
- **Output**
- **Library Name**
- **ASCII Technology File Name**
- **Scale UU/DBU**
- **Units**
- **Process Nice Value 0–20**
- **Error Message File**
User-Defined Data and Options are buttons that bring up other forms.

**User-Defined Data**

The *User-Defined Data* button displays the “Stream In User-Defined Data Form” on page 63. This form lets you specify optional mapping tables and user-defined data, such as the *User-Defined Property Mapping File*, *User-Defined Property Separator*, and *User-Defined SKILL File*.

**Options**

The *Options* button displays the *Stream In Options Form*. This form contains more parameters that help you specify how *Import – Stream* translates your Stream file and generates an error file.

**Set Fast Options**

Use the *Set Fast Options* button to set certain performance intensive options available in Stream In to their default values. These options are listed below:

- *Hierarchy Depth Limit*
- *Report Bad Polygons*
- *Precision Report*
- *Convert Array To Simple Mosaic*
- *Keep Pcells*
- *Rod Directory*
- *Maximum Vertices in Path/Polygon*
- *Convert Paths to Polygons*
- *Restore Pin Attribute*

During Stream In, PIPO performance might be negatively impacted with the values that you specify for these options. This happens because they require extra processing of data during translation. The negative impact on PIPO performance can be prevented by setting the default values of such options before importing Stream files.

When you use *Set Fast Options*, a dialog box appears and displays a warning message about overwriting the values you might have already specified with the default values of the Stream In options.

The *Set Fast Options* option does not update the template file unless you save it by using the *Save* option. For more information on PIPO performance, see *Guidelines for Stream In*. 
Template File

Use the Template File field to load filenames and option settings into the Virtuoso® Stream In form. When you use Import – Stream, a template file is optional. However, if you use the pipo command, you must create and specify a template file.

Load

The Load button updates the form with the option settings in the template file.

Save

The Save button writes the current option settings to the template file.

Browse

The Browse button opens a UNIX Browser window to let you search for the template file.

Run Directory

The Run Directory field takes the default directory for all the files you specify in the Virtuoso® Stream In form. If you want to read from or write to a file that is not in the run directory, type the full path in the field for that file.

Default: current directory (.)

Input File

Use the Input File field if you want to translate the Stream file. Alternatively, use the Browse button to search for the input file.

Top Cell Name

The Top Cell Name field takes the name of the cell at the top of the hierarchy that you want to translate, and the name of the view for the cells you translate. If you want to translate all cells in the Stream file to a layout view, leave the field empty. If you want the cells translated to a view other than layout, type the cell name and the view name in the field.
Default: layout

For example, if the cell hierarchy is as follows:

```
  A
 / \
B   C
 |   |
D   F
```

then the some examples of how you can translate cells of how the cells will be translated are as shown below:

<table>
<thead>
<tr>
<th>Cells to translate</th>
<th>Design Framework II view to translate into</th>
<th>What to type in the Top Cell Name field</th>
</tr>
</thead>
<tbody>
<tr>
<td>B F</td>
<td>layout</td>
<td>B</td>
</tr>
<tr>
<td>C D F E</td>
<td>layout3</td>
<td>C layout3</td>
</tr>
<tr>
<td>All cells</td>
<td>layout</td>
<td></td>
</tr>
<tr>
<td>All cells</td>
<td>layout3</td>
<td>* layout3</td>
</tr>
</tbody>
</table>

If you want Import – Stream to translate the cells in the hierarchy to different views, you need to create a cell name mapping file that specifies the cell names and views one by one. For more information about creating a cell mapping file, refer to the section “Cell Name Mapping File” on page 27.

**Output**

The **Output** option specifies the type of file Import – Stream generates.

**Opus DB**

The **Opus DB** option translates the Stream file into a Design Framework II database. You must specify a library name in the **Library Name** field for the Design Framework II database. You can also specify a technology file in the **ASCII Technology File Name** field for Import – Stream to use to compile the database. If you do not specify a technology file, Import – Stream uses the default technology file in the samples/techfile directory.

**ASCII Dump**
The ASCII Dump option translates the Stream file to ASCII format. If you select this option, you must specify a filename in the ASCII Dump File field. If you set this option on, Import – Stream does not generate a Design Framework II database.

**ASCII Dump File**

This ASCII Dump File option is available when you set the Output option to ASCII Dump. This is the file in which Import – Stream writes a text interpretation of the Design Framework II database.

**TechFile**

The Techfile option extracts information about the layers from the input Stream file and creates an ASCII technology file. Import – Stream does not generate a Design Framework II database when this option is selected. When you set this option on, you must type a filename in the ASCII Technology File Name field.

**Library Name**

The Library Name field takes the Design Framework II database library file in which Import – Stream writes the translated Stream file. When you set the Output option to Opus DB, Import – Stream uses this field. Library name can have a maximum of 255 characters. The Library Name option specifies the name of the library. The location of the library is specified by the Run Directory option. Even if the user gives a complete path of the library, only the library name is extracted.

For example, If the user specifies the complete path ./test_4/lib1, only lib1 will taken as library name and the rest of it will be ignored.

**Note:** PIPO Stream In cannot create a library if the library name specified by the user has more than 255 characters. In such a case, PIPO will issue an error and stop.

**ASCII Technology File Name**

Use the ASCII Technology File Name field to provide the name of the technology file you want to create when you set the Output option to TechFile. Alternatively, use the Browse button to search for the technology file. If you choose Output option Opus DB, Import – Stream uses the technology file you specify here to create the Design Framework II database.

**Scale UU/DBU**

Use the Scale UU/DBU field to take the ratio of database units per user unit you want for the Design Framework II database. Scale UU/DBU can change the precision of your design, but it cannot scale your design. For information on scaling your design, refer to the section “Scaling Your Design” on page 24.
Units

Units is the unit of measure in the Design Framework II file.

- **micron**
  The micron option sets the unit to one micron (0.001 millimeter).

- **millimeter**
  The millimeter option sets the unit to one millimeter.

- **mil**
  The mil option sets the unit to one mil (approximately 25.4 microns).

Process Nice Value 0-20

Use the Process Nice Value 0-20 option to adjust the nice priority value. The nice priority value is a factor used by the UNIX operating system to schedule the CPU time allocated to a process. The larger the value, the longer Import – Stream takes to translate your design.

Valid Values: an integer between 0 and 20, with 0 being the highest priority. Default: 0

Error Message File

Use the Error Message File field to write all warning, informational, and error messages that occur as it translates data during Import – Stream. Use the Browse button to search for the error message file. Default: PIPO.LOG.
Stream In User-Defined Data Form

Cell Name Map Table

Use the Cell Name Map Table field to map cell names from the Stream file to the Design Framework II database. The Cell Name Map Table option takes an optional file as input. You can also specify cell name mapping information in a SKILL file you specify in the User-Defined SKILL File text entry field at the bottom of this form. If you do not provide cell mapping information, Import – Stream uses the Case Sensitivity option setting from the “Stream In Options Form” on page 66. Cell names can have a maximum of 255 characters only. If input stream files have longer names, use the cell name map table to map long names to names with a maximum of 255 characters.

For a flow diagram of how Import – Stream determines how to translate a cell name and detailed information about how to create a cell name map table, refer to the section Cell Name Mapping File.

Note: The limit on each mapping entry in the cell map file has been enhanced to 1024 characters.
Caution

PIPO Stream In allows up to 255 characters in the STRNAME variable. If any STRNAME record in the input stream file has more than 255 characters, PIPO will issue an error and stop. In such a case, user is expected to supply mapping to long cell names that are up to 255 characters in size.

Layer Map Table

Use the Layer Map Table field to specify the Layer Name Mapping File that maps Stream layers to Design Framework II database layer-purpose pairs (LPP). It is optional to specify this file. Specifying this file can be useful when you want to change layer numbers, merge different layers into one layer, and filter layers.

You can also specify layer mapping information in a SKILL file you specify in the User-Defined SKILL File text entry field at the bottom of this form. If you specify a layer map file, Import – Stream does not check a SKILL file for layer mapping.

Text Font Map Table

Use the Text Font Map Table field to map Stream fonts and text sizes to Design Framework II fonts. The Text Font Map Table option takes an optional file as input. You can also specify text font mapping information in a SKILL file you specify in the User-Defined SKILL File text entry field at the bottom of this form. If you specify a text font map file, Import – Stream does not check a SKILL file for font mapping. For more information about how to create a text font map file, refer to the section “Text Font Mapping File for Stream In” on page 32.

Restore Pin Attribute

The Restore Pin Attribute field allows you to specify a nonzero value for the specified attribute number and restores the dumped connectivity information.

User-Defined Property Mapping File

The User-Defined Property Mapping File is an optional file you can use to map object types and property names from the Stream file to the Design Framework II database. If you do not specify a filename, Import – Stream tries to locate property name and value pairs in the Stream file by searching for equal signs (=) and a character separator (usually a comma [,]). You can specify the character Import – Stream uses to separate the property string with the User-Defined Property Separator option on this form.
The following table shows examples of how Import – Stream reads the Stream input when you do not specify a property mapping file:

<table>
<thead>
<tr>
<th>Stream Property</th>
<th>Design Framework II Property</th>
</tr>
</thead>
</table>
| 123 = "width=300,length=200" | width="300"
|                       | length="200"
| 124 = "single,length=200" | "STREAM PROPERTY 124" =
|                       | "single,length = 200"
| 124 = "param=100,alone,id=3" | param = "100"
|                       | "STREAM PROPERTY 124" =
|                       | "param = 100,alone,id=3"
| 125 = "output,SC/"    | "STREAM PROPERTY 125" = "output,SC/"

For more information about how to create a property map table, refer to the section “Property Mapping File for Stream In” on page 35.

**User-Defined Property Separator**

Use the User-Defined Property Separator field to separate properties. Most Stream files use a comma (,) to separate property values, but some files use other characters or spaces. If you specify a User-Defined Property Mapping File, Import – Stream ignores the Property Separator defined in the form. Default: a comma (,)

**User-Defined SKILL File**

The User-Defined SKILL File field takes a file containing user-defined SKILL routines that the system uses to translate cell names, layer names, text fonts, and illegal shapes. For information about writing SKILL routines, see the chapter “Using SKILL to Customize Stream and CIF Files” on page 269 or refer to the SKILL Language User Guide and SKILL Language Reference.
# Stream In Options Form

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report Bad Polygons</td>
<td></td>
</tr>
<tr>
<td>Snap XY to Grid Resolution</td>
<td></td>
</tr>
<tr>
<td>Convert Array to Simple Mosaic</td>
<td></td>
</tr>
<tr>
<td>Skip Undefined Layer-Purpose Pair</td>
<td></td>
</tr>
<tr>
<td>Convert Zero Width Paths to</td>
<td>lines</td>
</tr>
<tr>
<td>Case Sensitivity</td>
<td>preserve</td>
</tr>
<tr>
<td>Text Case Sensitivity</td>
<td>preserve</td>
</tr>
<tr>
<td>Convert Nodes to</td>
<td></td>
</tr>
<tr>
<td>Keep PCells</td>
<td></td>
</tr>
<tr>
<td>Replace [] with &lt;&gt;</td>
<td></td>
</tr>
<tr>
<td>Merge Undefined Purpose to drawing</td>
<td></td>
</tr>
<tr>
<td>Precision Report</td>
<td></td>
</tr>
<tr>
<td>Ignore BOX Record</td>
<td></td>
</tr>
<tr>
<td>Retain Reference Library (No Merge)</td>
<td></td>
</tr>
<tr>
<td>Do Not Overwrite Existing Cell</td>
<td></td>
</tr>
<tr>
<td>Filter Out Warning/Information Messages</td>
<td></td>
</tr>
<tr>
<td>Filter Out Unmapping Warning</td>
<td></td>
</tr>
<tr>
<td>Hierarchy Depth Limit</td>
<td>32</td>
</tr>
<tr>
<td>Maximum Vertices in Path/Polygon</td>
<td>1024</td>
</tr>
<tr>
<td>Rod Directory</td>
<td></td>
</tr>
<tr>
<td>Reference Library Order</td>
<td></td>
</tr>
<tr>
<td>Keep Stream Cells</td>
<td></td>
</tr>
<tr>
<td>Attach Techfile Of Library</td>
<td></td>
</tr>
<tr>
<td>Comprehensive Log</td>
<td></td>
</tr>
<tr>
<td>Ignore Pcell evaluation failure</td>
<td></td>
</tr>
<tr>
<td>Append into existing Database</td>
<td></td>
</tr>
<tr>
<td>Generate Hierarchy Listing</td>
<td></td>
</tr>
</tbody>
</table>
Report Bad Polygons

The Report Bad Polygon option determines whether Import – Stream removes collinear and coincident points in polygons and paths and reports incomplete or illegal polygons in the error file. A polygon can have more than one error, although Import – Stream reports only the most severe problem in the error file. Import – Stream reports the following problems, listed here in order of severity with the most severe problem described first.

<table>
<thead>
<tr>
<th>Open</th>
<th>An open polygon is one whose first and last coordinates do not coincide.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reentrant</td>
<td>A reentrant polygon is one in which the sum of the internal angles is not 360 degrees, or one in which one or more pairs of edges intersect rather than meet.</td>
</tr>
<tr>
<td>Acute angle</td>
<td>An acute angle is an angle that is smaller than 90 degrees.</td>
</tr>
<tr>
<td>Non-45 degree angle</td>
<td>A non-45 degree angle is an angle that is not a multiple of 45 degrees.</td>
</tr>
</tbody>
</table>

Valid Values: 90, 135, 180, 225, 270, 315, 360

Note: An angle of a polygon refers to the internal angle of the polygon.

Snap XY to Grid Resolution

The Snap XY to Grid Resolution option adjusts the coordinates of geometric objects to coincide with the grid resolution. When you use this option, you can use graphic commands (such as the Virtuoso Layout Editor commands Move, Stretch, and Split) to access end points, edges, and vertexes of geometric objects in your translated designs. However, this option can shift data from the original Stream grid, resulting in a loss of precision.

Default: off

Note: You can set the grid resolution using mfgGridResolution in the Physical Rules class of the technology file you specify in the ASCII Technology File field. If you do not define mfgGridResolution, Import – Stream sets it to the inverse of the value you set for the Scale UU/DBU field.

Convert Array to Simple Mosaic

The Convert Array to Simple Mosaic option converts each Stream array (AREF) to a single Design Framework II mosaic. If you turn this option off, each element in the AREF translates to an individual Design Framework II cell instance. Default: on

Skip Undefined Layer-Purpose Pair
The *Skip Undefined Layer-Purpose Pair* option ignores the layer-purpose pairs you have not defined in the technology file for the current design library. You can use this option to drop certain layers of the input data in the translation. If you do not have write access to the current design library’s technology file, Stream In issues a warning and automatically turns this option on.

**Default:** off

### Convert Zero Width Paths to

The *Convert Zero Width Paths to* option determines whether *Import – Stream* translates zero-width paths to lines.

- **lines**
  - The *lines* option translates zero-width paths to lines.
- **ignore**
  - The *ignore* option does not translate zero-width paths to the Design Framework II database.

### Case Sensitivity

The *Case Sensitivity* option specifies whether *Import – Stream* changes the case of letters in cell names. *Import – Stream* uses this option when you do not supply cell name mapping information for the cell. You can map cell names in a cell name map table or in a SKILL file.

- **preserve**
  - The *preserve* option translates cell names without changing case. For example, the name *Obj1* remains *Obj1*.
- **upper**
  - The *upper* option translates all cell names to uppercase. For example, the name *Obj1* becomes *OBJ1*.
- **lower**
  - The *lower* option translates all cell names to lowercase. For example, the name *Obj1* becomes *obj1*.

### Text Case Sensitivity

The *Text Case Sensitivity* option specifies whether *Import – Stream* changes the case of letters in text labels. *Import – Stream* uses this option when you do not supply text label
mapping information for the cell. You can map text labels in a SKILL file by using the `piTextMap()` function.

**Note:** If you specify the mapping information using the `piTextMap` function, the information specified in the *Text Case Sensitivity* option is ignored.

**preserve**

The *preserve* option translates text labels without changing case. For example, the label `Obj1` remains `Obj1`.

**upper**

The *upper* option translates the text label to uppercase. For example, the label `Obj1` becomes `OBJ1`.

**lower**

The *lower* option translates the text labels to lowercase. For example, the label `Obj1` becomes `obj1`.

**Convert Nodes to**

The *Convert Nodes to* option determines whether Import – Stream translates nodes from the stream file to dots in the CDBA.

**dots**

The *dots* option translates nodes to dots in CDBA. The dots are drawn at the same coordinates as the nodes. Any properties on the nodes are also translated to dot properties if the property names are specified in the property mapping file. The object type in the property mapping file must be specified as `dot`.

**ignore**

The *ignore* option does not translate nodes to dots in CDBA and ignores them.

**Keep PCells**

The *Keep PCells* option preserves parameterized cells by reading the SKILL files already created by Stream Out in the `KPDIR` directory located in the run directory.

If the *Keep PCells* option is selected, Stream In will discard the pcell description contained in the Stream file and replace it in the dfII database by loading in the corresponding pcell SKILL files that are dumped during Stream Out in the `KPDIR` directory.

You can read about the same option on the *Stream Out Options Form* on page 106.
Important

Stream In requires a correct technology library for preserving pcells. The technology library can be specified by either the Attach Techfile of Library option (in the Stream In Options form) or the attachTechfileOfLib option (in the template file). Therefore, if either you do not specify a technology library while using the Keep PCells option or the specified technology library is corrupt, Stream In will generate the following message and stop:

“FATAL (59): Failed to load technology file KPDIR/kp_techfile_devices.tf. This file might be corrupt. Recreate it using Stream Out.”

Additionally, the technology file of the target library must be complete for the successful loading of the pcell SKILL files. This implies that at least the layer definitions and layer rules sections of the target library technology file must be super sets of the layer definitions and layer rules sections of the original source library.

Caution

Because pcells will be recreated simply by loading SKILL files, the Layer Map Table will be ignored for the objects in these pcells recreated during Stream In.

Note: If you use the Keep Pcells option in conjunction with the Retain Reference Library (No Merge) option during stream in, the Keep Pcells option takes precedence over the Retain Reference Library (No Merge) option. Therefore, when both the options are used together, PIPO uses the pcell files in KPDIR that are created during stream out and does not retain the relationship to the reference libraries.

Replace [ ] with <>

The Replace [ ] with <> option replaces the character [ with < and ] with > in text records during the translation from the Stream file to a dfII library.

Default: off

Merge Undefine Purpose to drawing

The Merge Undefine Purpose to drawing option maps all Stream datatypes to Cadence purpose drawing.

Precision Report
The **Precision Report** option indicates whether *Import – Stream* checks precision and it writes a warning message to the error file if precision is lost during translation. The loss of precision occurs when *Import – Stream* rounds coordinate values and a shape falls off grid. When this option is selected, *Import – Stream* checks precision and writes warnings to the error file, and translation is slower. When this option is deselected, *Import – Stream* does not check precision.

**Default:** *off*

### Ignore BOX Record

The **Ignore BOX Record** option indicates whether *Import – Stream* translates BOX records. A BOX record is an alternate way of defining a rectangular shape. The record consists of a coordinate for the center of the shape and values for the height and width of the shape. When this option is deselected, *Import – Stream* translates each BOX record into the lower left and upper right coordinates of the shape. When selected, *Import – Stream* ignores BOX records.

**Default:** *off*

### Retain Reference Library (No Merge)

The **Retain Reference Library (No Merge)** option indicates whether the Stream file retains the reference library relationship for cells that are not defined in the current library. If you use this option while creating the Stream file using *Export – Stream*, only the cell name and placement information will be stored in the Stream file. For more information about translating reference libraries, refer to the section “Translating Reference Libraries Using Stream” on page 48.

**Important**

If you use the **Retain Reference Library (No Merge)** option in conjunction with the **Keep Pcells** option during stream in, the **Keep Pcells** option takes precedence over the **Retain Reference Library (No Merge)** option. Therefore, when both the options are used together, PIPO uses the pcell files in the *KPDIR* that are created during stream out and does not retain the relationship to the reference libraries. The **Retain Reference Library (No Merge)** option, in such case, does not impact how the pcells are streamed in.

**Note:** When the **Retain Reference Library (No Merge)** option is used during Stream In, the Stream file has only the cell name and placement information, but the reference library relationship needs to be maintained. The reference libraries must exist in the *cds.lib* file during Stream In. Typically, select this option both during Stream Out and Stream In if you require the stream creation and the library generation processes to happen quickly, and the complete design information is not needed (consider application such as the translation of
abstracts). Avoid using this option in both Stream Out and Stream In if complete design information is required in a Stream file (for running LVS on a complete design). You might want to try more permutations after referring to the details in the section “Translating Reference Libraries Using Stream” on page 48.

Do Not Overwrite Existing Cell

The Do Not Overwrite Existing Cell option prevents any cell that already exists in the library from being overwritten.

Default: off

Filter out Warning/Information Messages

The Filter out Warning/Information Messages option indicates whether Import – Stream writes warning and information messages to the error file during translation. If you turn this option on, Import – Stream writes only error messages to the error file. If you turn this option off, Import – Stream writes all messages to the error file.

Default: off

Filter Out Unmapping Warning

The Filter Out UnMapping Warning option lets you choose whether Import-Stream writes warning messages about Cadence layers to the error file during the translation.Warnings about user layers are not affected.

Default: off

If you select this option, the following warnings will be filtered:

- **Message ID: 335**

  Stream layer rule for the layer-purpose pair `strmLayer:strmType` is nil. All objects on this LPP will be dropped.

- **Message ID: 408**

  Stream layer-datatype pair `layerNumber:purposeNumber` is not defined in the layer map file. This layer-purpose pair will be ignored.

Hierarchy Depth Limit

Use the Hierarchy Depth Limit field to specify the depth of the design hierarchy to be translated during Stream In. Stream In stops the translation process at the hierarchy level indicated by this option. Default: 32
**Note:** If the hierarchy depth of the input design is more than the value of the *Hierarchy Depth Limit* option, then one time warning message is displayed.

**Maximum Vertices in Path/Polygon**

The *Maximum Vertices in Path/Polygon* field specifies the maximum number of vertexes you want in polygons and paths in dfII. *Import – Stream* divides polygons and paths with more than this number of vertexes into smaller objects. Default: 1024

Valid Values: a positive integer from 4 to 4000

**Note:** Though the default is set to 1024, the maximum limit for vertexes in dfII is 4000. The maximum limit imposed by the GDSII format is 200 and that by the Stream format is 8000. The Stream/GDSII reader writer of each EDA vendor is therefore responsible for handling restrictions imposed by their databases with this option.

**Caution**

*This option is intended for use with designs containing polygons with very large numbers of vertexes, for example, 4000 or more. If you set this option to a value that causes Import – Stream to divide Stream polygons into many dfII polygons, performance is reduced considerably and in some cases the translation fails.*

**Rod Directory**

The *Rod Directory* field specifies the location of the ROD directory for the PIPO-ROD interface to work. When you specify a ROD directory as input to the Cadence Stream reader, Stream uses both the Stream Property 126 data and the auxiliary ROD files in the ROD directory to recreate ROD-specific data.

Information on using Stream In and Stream Out translators is published on [SourceLink](#).

**Caution**

*If you stream in ROD data without the auxiliary ROD files or use a non-Cadence Stream reader, only geometric data is streamed. ROD data is not recreated in the streamed -in design.*

**Reference Library Order**

The *Reference Library Order* field specifies a comma, space, or tab separated ordered list of reference library names. While looking for master cellviews of instances, libraries are picked up for search in the order that is specified. When no names are specified, all the libraries listed in the *cds.lib* file are searched. However, the search order in such a case is
undefined.
Example: basic myLib sample

It is important to note the following:

- If any library names are specified in this list then other libraries listed in cds.lib are not used. Therefore, it is important to list either no library names (to allow the use of cds.lib) or list all the reference library names needed by this input stream file.

- Only library names can be specified. Their paths are still taken from the cds.lib files.

- Each library name specified in this list must also be present in the cds.lib file.

- The output (target/destination) library name can also be specified. If it is not present in the list then internally, it is automatically assigned as the first library in the search order.

The Reference Library Order option is useful by:

- Enforcing an order of search for masters of instances.

- Limiting the set of libraries for searching, thus making the search faster.

Keep Stream Cells

The Keep Stream Cells option provides a way of referring to the information pertaining to a particular cell in the stream file first instead of a reference library. While creating an instance during Stream In, PIPO will not accept information in the stream file by default for a particular cell if the cell is present in the reference library. If the value of the Keep Stream Cells option is set to true, while creating instances, the stream file will be searched before the reference library or the design libraries. If the value is nil or "", then the current default behavior of PIPO will be followed; the stream file will not be searched while creating instances.

Note: The Do Not Overwrite Existing Cell option overrides the Keep Stream Cells option when the cell being translated already exists in the target library.

Attach Techfile of Library

The Attach Techfile of Library field lets you to use and attach an existing technology file to the target library.

Specify the name of the library that is to be attached in the Attach Techfile Of Library field. Also specify the path to the technology library in the cds.lib file.

When you use this option, PIPO does not create a new technology file but uses the one from the library specified in the Attach Techfile Of Library option. It does not modify the existing technology file even if it is writable. Any new layer-purpose pairs (layers that are not defined in the technology file) from the stream file are ignored.
If you have also specified an ASCII technology file in the *ASCII Technology File Name* option, it will be ignored. Instead, the technology file specified in this option will be used for creating the dfII database.

An exception to the rule:

If the target library already contains a technology file, then the library of the technology file specified in the *Attach Techfile Of Library* option will be ignored.

**Note:** It is assumed that the technology file provided for attachment is complete.

### Comprehensive Log

Select the *Comprehensive Log* option to output the warnings and error messages generated by SKILL function calls and PIPO dependencies into the PIPO log file.

Default: nil

### Ignore Pcell Evaluation Failure

Select the *Ignore Pcell Evaluation Failure* option if you do not want PIPO to halt upon any pcell evaluation failure. If you do not select this option, PIPO will halt upon any pcell evaluation failure and a fatal message will be displayed.

Default: nil

### Append into existing Database

Use the *Append into existing Database* option to append the contents of cellviews into dfII without overwriting the existing data. Set this option to true before starting Stream In.

Default: False

**Note:** Do not set both the *Do not Overwrite Existing Cell* and the *Append into Existing Database* options to TRUE at the same time otherwise a fatal message will be generated.

### Generate Hierarchy Listing

The *Generate Hierarchy Listing* option writes the hierarchical information in the log file. This is a boolean option. For example, when this option is selected, following information related to the top cells and hierarchy will be written to the PIPO log file.

```
Top Cells in File
----------------
TOP layout

List Hierarchy(#cellInst,#arrayInst)
-----------------------------------
```
Translating Stream Files

TOP layout
  .CELL4 layout(1)
  ..CELL3 layout(3)
  ...CELL2 layout(10)
  ...CELL1 layout(5)

Default: nil

Translating Compressed Stream File

Stream In provides the feature of translating the compressed Stream file. Supported compression utilities are `gzip`, `bzip2`, and `compress`. If a compressed file has `.gz` or `.GZ` extension, `gzip` utility is used to uncompress the file. If the file extension is `.bz` or `.bz2`, files are uncompressed using the `bzip2` utility. Incase of `.Z`, files are uncompressed using the `uncompress` command.

Based on the file extension, PIPO assumes that it is a compressed file. If Stream In does not find the compression utility based upon the file extension, it displays a fatal message. For example, if the file `out.gds.gz` is Streamed in and the compression utility `gzip` does not exist, then a fatal message is displayed. For more information about the fatal and warning messages, see Appendix B, “PIPO Messages”.

Starting Import – Stream from a Form


   The Virtuoso® Stream In form appears.

2. Type the name of the input Stream file in the Input File field.

3. Type the library name where you want to place the translated design into the Library Name field.

4. Specify any optional values that you want on the Stream In Options form.

5. Click Apply or OK on the forms.

   Import – Stream begins the conversion. You can check in the CIW for the job number and for information about whether the translation succeeded.
At the end of execution, a pop-up box shows the status of your runs. Here is an example of such a pop-up box:

![Pop-Up Message](image)

The above displayed pop-up box contains the job number of your run. For example, (PID = ipc:1). It also gives you the status of your run along with number of errors/warnings if any.

The *Display Log* button displays the contents of the log file.

### Starting Import – Stream from the Command Line

1. Create a template file by using the *Virtuoso® Stream In* form.
   
   For information about how to create the template file, see the section “Preparing a Template File and .cdsinit File for Stream In” on page 79 below.

2. To generate a technology file for *Import – Stream*, type the following command:
   
   ```bash
   pipo strmtechgen templateFile
   ```

3. To translate the Stream file to a Design Framework II database, type the following command:
   
   ```bash
   pipo strmin templateFile
   ```

### Stream In GUI and Template File Options

The following table provides a list of all the Stream In GUI options and their corresponding names in the template file.

<table>
<thead>
<tr>
<th>Stream In GUI</th>
<th>Template File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Directory</td>
<td>runDir</td>
</tr>
</tbody>
</table>
### Stream In GUI

<table>
<thead>
<tr>
<th><strong>Stream In GUI</strong></th>
<th><strong>Template File</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Input File</td>
<td>inFile</td>
</tr>
<tr>
<td>Top Cell Name</td>
<td>primaryCell</td>
</tr>
<tr>
<td>Library Name</td>
<td>libName</td>
</tr>
<tr>
<td>ASCII Technology File Name</td>
<td>techfileName</td>
</tr>
<tr>
<td>Scale UU/DBU</td>
<td>scale</td>
</tr>
<tr>
<td>Units</td>
<td>units</td>
</tr>
<tr>
<td>Error Message File</td>
<td>errFile</td>
</tr>
<tr>
<td>Retain Reference Library (No Merge)</td>
<td>refLib</td>
</tr>
<tr>
<td>Hierarchy Depth Limit</td>
<td>hierDepth</td>
</tr>
<tr>
<td>Maximum Vertices in Path/Polygon</td>
<td>maxVertices</td>
</tr>
<tr>
<td>Report Bad Polygons</td>
<td>checkPolygon</td>
</tr>
<tr>
<td>Snap XY to Grid Resolution</td>
<td>snapToGrid</td>
</tr>
<tr>
<td>Convert Array to Simple Mosaic</td>
<td>arrayToSimMosaic</td>
</tr>
<tr>
<td>Text Case Sensitivity</td>
<td>caseSensitivity</td>
</tr>
<tr>
<td>Convert Zero Width Paths to</td>
<td>zeroPathToLine</td>
</tr>
<tr>
<td>The preserve option translates text labels without changing case. For example, the label <code>Obj1</code> remains <code>Obj1</code>.</td>
<td>convertNode</td>
</tr>
<tr>
<td>Keep PCells</td>
<td>keepPcell</td>
</tr>
<tr>
<td>Replace [] with &lt;&gt;</td>
<td>replaceBusBitChar</td>
</tr>
<tr>
<td>Skip Undefined Layer-Purpose Pair</td>
<td>skipUndefinedLPP</td>
</tr>
<tr>
<td>Ignore BOX Record</td>
<td>ignoreBox</td>
</tr>
<tr>
<td>Merge Undefined Purpose to drawing</td>
<td>mergeUndefPurposToDrawing</td>
</tr>
<tr>
<td>Precision Report</td>
<td>reportPrecision</td>
</tr>
<tr>
<td>Keep Stream Cells</td>
<td>keepStreamCells</td>
</tr>
<tr>
<td>Attach Techfile of Library</td>
<td>attachTechfileOfLib</td>
</tr>
<tr>
<td>Filter out Warning/Information Messages</td>
<td>runQuiet</td>
</tr>
<tr>
<td>Do Not Overwrite Existing Cell</td>
<td>noWriteExistCell</td>
</tr>
</tbody>
</table>
Design Data Translator's Reference
Translating Stream Files

<table>
<thead>
<tr>
<th>Stream In GUI</th>
<th>Template File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter Out Unmapping Warning</td>
<td>NOUnmappingLayerWarning</td>
</tr>
<tr>
<td>Ignore Pcell Evaluation Failure</td>
<td>ignorePcellEvalFail</td>
</tr>
<tr>
<td>Cell Name Map Table</td>
<td>cellMapTable</td>
</tr>
<tr>
<td>Layer Map Table</td>
<td>layerTable</td>
</tr>
<tr>
<td>Text Font Map Table</td>
<td>textFontTable</td>
</tr>
<tr>
<td>Restore Pin Attribute</td>
<td>restorePin</td>
</tr>
<tr>
<td>User-Defined Property Mapping File</td>
<td>propMapTable</td>
</tr>
<tr>
<td>User-Defined Property Separator</td>
<td>propSeparator</td>
</tr>
<tr>
<td>User-Defined SKILL File</td>
<td>userSkillFile</td>
</tr>
<tr>
<td>Rod Directory</td>
<td>rodDir</td>
</tr>
<tr>
<td>Reference Library Order</td>
<td>refLibOrder</td>
</tr>
<tr>
<td>Comprehensive Log</td>
<td>comprehensiveLog</td>
</tr>
<tr>
<td>Append into existing Database</td>
<td>appendDB</td>
</tr>
<tr>
<td>Generate Hierarchy Listing</td>
<td>genListHier</td>
</tr>
</tbody>
</table>

Preparing a Template File and .cdsinit File for Stream In

You can define a set of option values for loading into the Virtuoso® Stream In form in a template file or in the .cdsinit file. The values required in the Stream In template file and the .cdsinit file are the same. You can load a template file into the Virtuoso® Stream In form when you use Import – Stream or pipo. (The pipo command is the non-interactive, batch mode version of Import – Stream and Export – Stream.) The system automatically loads the .cdsinit file when you start the software. You can create a template file in one of two ways:

- Enter values in the Virtuoso® Stream In, Stream In Options, and Stream In User-Defined Data forms and click Save to save the option values to the file you specify as the template file.
- Create a text file. You can copy the sample template file streamIn.il in the samples/transUI directory and modify the file.
The required values in the Stream In template file are the name of the Stream-formatted input file and the name of the Design Framework II library. The other values are optional. The following is a sample Stream In template file:
Design Data Translator's Reference
Translating Stream Files

;***************************************************************
; Cadence Design Systems Inc. All Rights Reserved.           *
;***************************************************************

;***************************************************************
; filename : streamIn.il
;***************************************************************
streamInKeys = list(nil
'runDir                  "; Run Directory
'inFile                  "test.gds"; - Name of input Stream file - Input File
'primaryCell             "" ;- Top Cell Name
'libName                 "strmLib" ;- Library Name
'techfileName            "" ;- ASCII Technology File Name
'scale                   0.001000 ; Scale UU/DBU
'units                   "micron" ; "micron" "milimeter" / "mil" - Units
'errFile                 "PIPO.LOG"; - Error Message File
'reflib nil ;- t/nil - Retain Reference Library (No Merge)
'hierDepth               32 ; (0-32) - Hierarchy Depth Limit
'maxVertices 1024 ; (1-1024) - Maximum Vertices in Path/Polygon
'checkPolygon nil ;- nil - Report Bad Polygons
'snapToGrid nil ;- nil - Snap XY to Grid Resolution
'arrayToSimMosaic t ;- nil - Convert Array to Simple Mosaic
'caseSensitivity "lower" ; "preserve"/"lower"/"upper" - Case Sensitivity
'zeroPathToLine "lines" ; "lines"/"ignore" - Convert Zero Width Paths to
'convertNode "Dot" ; "dots"/"ignore" - Convert Nodes to
'keepPcell nil ;- t/nil - Keep PCells
'replaceBusBitChar nil ;- t/nil - Replace [ ] with <>
'skipUndefinedLPP nil ;- t/nil - Skip Undefined Layer-Purpose Pair
'ignoreBox nil ;- nil - Ignore BOX Record
'mergeUndefinedPurposeToDrawing nil ;- t/nil - Merge Undefined Purpose to drawing
'reportPrecision nil ;- t/nil - Precision Report
'keepStreamCells nil ;- t/nil - Keep Stream Cells
'attachTechfileOfLib "" ;- Attach Techfile of Library
'runQuiet nil ;- t/nil - Filter out Warning/Information Messages
'noWriteExistCell nil ;- t/nil - Do Not Overwrite Existing Cell
'NOUnmappingLayerWarning nil ;- t/nil - Filter Out Unmapping Warning
'comprehensiveLog nil ;- nil - Comprehensive Log
'ignorePcellEvalFail nil ;- t/nil - Ignore Pcell Evaluation Failure
'appendDB nil ;- t/nil - Append into Existing Database
'cellMapTable "" ;- Cell Name Map Table
'layerTable "layerMap"; - Layer Map Table
'textFontTable "" ;- Text Font Map Table
'restorePin 0 ;(0-127) - Restore Pin Attribute
'propMapTable "" ;- User-Defined Property Mapping File
'propSeparator "" ;- User-Defined Property Separator
'userSkillFile "" ;- User-Defined SKILL File
'rodDir "" ;- Rod Directory
'reflibOrder "" ;- Reference Library Order
'genListHier nil ;- Generate Hierarchy Listing
)
For detailed information on the above options, refer to the Import – Stream Forms section.

**Note:** Set the ‘gdsCompliantMosaic’ option to t in the Stream In template file to indicate to PIPO that the input stream file contains mosaics in the standard GDSII stream format as follows:

```
gdsCompliantMosaic nil ;t,nil
```

When this option is set to nil, you might find some placement differences in the way PIPO Stream In translates or rotates mosaics to be compliant with the Stream Out process. Default: nil.

## How Import – Stream Translates Data

There are important differences between the GDSII Stream format and the Design Framework II format. This section summarizes the major differences. To ensure successful translation of your data, familiarize yourself with the format differences listed in this section. For a full description of the GDSII Stream format, refer to Appendix A, “Stream Format”.

The major differences between Stream format and Design Framework II format are described below:

- Stream permits TEXT, AREF, and SREF magnification. dfII format supports only TEXT and SREF magnification.

- Stream permits more than 4000 points in a polygon. dfII format limits a polygon to a maximum of 4000 points. Import – Stream cuts polygons with more than 4000 points to smaller polygons and writes an information message to the error file.

- Stream supports absolute angles. dfII format does not. Import – Stream converts only relative-angle-placed text, instances, and arrays; it discards information dealing with absolute angles and writes a warning message to the error file.

Stream permits rotation at any angle. dfII format supports only 90-degree rotations. Import – Stream translates Stream angle rotations into manhattan-placed text, instances, and arrays.

The following table shows the ranges of Stream angle rotations and the Design Framework II rotations that result when you use Import – Stream.

<table>
<thead>
<tr>
<th>Stream rotation angle</th>
<th>Stream reflection</th>
<th>dfII format</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ≤ and ≤ 45</td>
<td>yes</td>
<td>MX</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>R0</td>
</tr>
<tr>
<td>&gt; 45 and ≤ 135</td>
<td>yes</td>
<td>MXR90</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>R90</td>
</tr>
</tbody>
</table>
Design Framework II format does not support zero-width paths. Import – Stream translates zero-width paths into lines.

Stream supports path types and widths on text. Design Framework II format supports only path type 0 and path width 0 on labels. Import – Stream does not translate the path width of a label to Design Framework II format.

Because of the structural similarities between AREFs and simple arrays, Import – Stream can translate AREFs directly to simple arrays. However, if you turn off the Convert Array to Simple Mosaic option on the Stream In Options form, Import – Stream translates every element in an AREF to an individual instance.

The following table shows how path types map between the Stream format and the dfII format.

<table>
<thead>
<tr>
<th>Stream rotation angle</th>
<th>Stream reflection</th>
<th>dfII format</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;135 and ≤225</td>
<td>yes</td>
<td>MY</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>R180</td>
</tr>
<tr>
<td>&gt;225 and ≤315</td>
<td>yes</td>
<td>MYR90</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>R270</td>
</tr>
<tr>
<td>&gt;315 and ≤360</td>
<td>yes</td>
<td>MX</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>R0</td>
</tr>
</tbody>
</table>

The following table shows how data types map between Stream format and Design Framework II.

<table>
<thead>
<tr>
<th>Stream data type</th>
<th>dfII data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>library</td>
<td>library</td>
</tr>
<tr>
<td>structure</td>
<td>cellview</td>
</tr>
</tbody>
</table>
### Import – Stream

- Import – Stream accepts more than 49 record types but does not map them to any Design Framework II object.

- GDSII version 6 supports 59 record types. Design Framework II supports up to 49.

The following record types are used for CustomPlus™ Stream files only:

- 48 (BGNEXTN)
- 49 (ENDEXTN)
- 52 (STRCLASS)

The following record types are for the multiple-reel Stream file and the format type of a Stream tape:

- 51 (TAPENUM)
- 52 (STRCLASS)
- 53 (RESERVED)
- 54 (FORMAT)
- 55 (MASK)
- 56 (ENDMASKS)

### Import – Stream Output Files

Depending on what optional files you specify in the Virtuoso® Stream In form or template file, Import – Stream produces one or more of the following files:

- Translated dfII database, with input objects mapped to output objects
- A text version of the Stream file
  - When Import – Stream generates this file, it does not translate the Stream file to Design Framework II format.
- An ASCII technology file
  - When Import – Stream generates this file, it does not translate the Stream file to Design Framework II format.
- An error file (called PIPO.LOG by default) that reports statistics and any warnings that might have occurred during translation
Error File

The following is a sample of an Import – Stream error message file or log file. The Import – Stream parameter values used to produce this example are included at the end of the file.

**************************************************************************
**   CADENCE Design Systems, Inc.  *
**   Virtuoso(R) STREAM In EXP   *
**   @(#)CDS: pipo version 5.0 10/18/2001 21:12 (cicser7s) $   *
**************************************************************************
**
Stream file    : /usr1/234753/Stream.1.out

Reading Stream File ...
VERSION       : 5
MODIFICATION  : Fri Jun  4 14:08:12 1999
GDS LIBRARY   : TERRYM.DB
U-UNIT/DBU    : 0.001000000000
METRIC/DBU    : 0.000000001000

TARGET LIBRARY PATH: /usr1/234753/rt_terrym

1. scanning cellview (a2a_gteq_64 layout)
2. scanning cellview (a2a_lukahd4 layout)
3. scanning cellview (a2a_lukahd4_3 layout)
4. scanning cellview (pcon layout)
5. scanning cellview (m12py layout)
6. scanning cellview (pcon1 layout)
7. scanning cellview (m12py2 layout)
PASS 2...
  1. translating cellview (a2a_gteq_64 layout)
  2. translating cellview (a2a_lukahd4 layout)
INFO (122): Creating a new layer-purpose pair 'L42 42 drawing' in the technology file.
INFO (122): Creating a new layer-purpose pair 'L37 37 drawing' in the technology file.
INFO (122): Creating a new layer-purpose pair 'L50 50 drawing' in the technology file.
INFO (122): Creating a new layer-purpose pair 'L1 1 drawing' in the technology file.
INFO (122): Creating a new layer-purpose pair 'L2 2 drawing' in the technology file.
INFO (122): Creating a new layer-purpose pair 'L51 51 drawing' in the technology file.
INFO (122): Creating a new layer-purpose pair 'L51 51 drawing' in the technology file.
INFO (122): Creating a new layer-purpose pair 'L60 60 drawing' in the technology file.
  3. translating cellview (a2a_lukahd4_3 layout)
  4. translating cellview (pcon layout)
  5. translating cellview (m12py layout)
  6. translating cellview (pcon1 layout)
  7. translating cellview (m12py2 layout)
Top Cells in File

a2a_gteq_64  layout
List Hierarchy (#cellInst,#arrayInst)

a2a_gteq_64  layout
..a2a_lukahd4_3 layout(16)
..pcon1 layout(11)
..m12py layout(6)
..m12py2 layout(1)
..pcon layout(1)
..a2a_lukahd4 layout(4)
..pcon1 layout(11)
..m12py layout(6)
..pcon layout(2)

Individual Cell Statistics

<table>
<thead>
<tr>
<th>CellView</th>
<th>Rectangles</th>
<th>Polygons</th>
<th>Paths</th>
<th>Ellipses</th>
<th>Labels</th>
<th>Instances</th>
<th>Arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dots</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Statistics Of Layers

<table>
<thead>
<tr>
<th>Layer #</th>
<th>Rectangles</th>
<th>Polygons</th>
<th>Paths</th>
<th>Ellipses</th>
<th>Labels</th>
<th>Dots</th>
</tr>
</thead>
</table>
Summary Of Objects
---------------------
1396 Rectangles
0 Polygons
326 Paths
0 Ellipses
24 Labels
58 Instances
0 Arrays
7 Cells
0 Dots

00:00:04 Elapsed time
00:00:03 CPU time

29485 Kbytes of memory

*** There were no error or warning message ***

The setup file’s contents :

streamInKeys = list(nil
    'runDir       ".",
    'inFile       "Stream.1.out",
    'primaryCell  "A2A_GTEQ_64",
    'libName      "rt_terrym",
    'techfileName "",
    'scale        '0.001000
    'units        "micron",
    'errFile      "PIPO.LOG.2.out",
    'refLib       nil
    'hierDepth    32
    'maxVertices  1024
    'checkPolygon nil
    'snapToGrid   nil
    'arrayToSimMosaic t
    'caseSensitivity "lower"
    'zeroPathToLine "lines"
    'skipUndefinedLPP nil
    'ignoreBox    nil
    'reportPrecision nil
    'runQuiet     nil
    'noWriteExistCell nil
    'NOUnmappingLayerWarning nil
    'cellMapTable  "cellmap.tbl.in"
    'layerTable   ""
    'textFontTable ""
    'restorePin   0
    'propMapTable ""
    'propSeparator ",,"
    'userSkillFile ""
    'rodDir       ""
ASCII Dump File

If you select ASCII Dump as the Output option in the Virtuoso® Stream In form, Import – Stream produces an ASCII image file of the input Stream file and does not translate the input file. The following is a sample ASCII dump file:

Begin Library
Library Name: main_lib, DB unit per user unit: 1000,
User Unit: Micron
Begin Cell Definition
Cell Name : chip_top, View Name : layout
Path-Layer: 3 Data Type: 0 No of points: 4 Width: 2000
Path Type: Truncate
(-2000,-500) (-2000,7000) (5000,7000) (5000,2000)
Polygon - Layer : 2 Data Type : 0 No of points : 9
(-5000,5000) (-9500,5000) (-9500,2500) (-8000,2500) (-8000,4000)
(-6500,4000) (-6500,3000) (-5000,3000) (-5000,500)
Rectangle - Layer : 1 Data Type : 0 BBOX : (1000,1000) (3000,3000)
Cell Instance - Master Name : block1 Inst Name :
Origin: (9000,5500) Angle: 0.000 Mirror: 0 Magnification: 1.000
Array - Master Name : block2 Array Name :
Origin: (-15000,-18500) Mirror: 0 Angle: 0.000
Magnification: 1.000
Rows : 2 Columns : 3 Row Spacing : 8500 Column Spacing : 15500
pcol : (31500,-18500) prow : (-15000,-1500)
End Cell Definition
Begin Cell Definition
Cell Name : block1, View Name : layout
Path-Layer: 3 Data Type: 0 No of points: 4 Width: 2000
Path Type: Truncate
(7500,0) (7500,7500) (14500,7500) (14500,2500)
Polygon - Layer : 2 Data Type : 0 No of points : 9
(4500,1000) (0,1000) (0,3000) (1500,3000) (1500,4500)
(3000,4500) (3000,3500) (4500,3500) (4500,1000)
Rectangle - Layer : 1 Data Type : 0 BBOX :
(10500,1500) (12500,3500)
End Cell Definition
Begin Cell Definition
Cell Name : block2, View Name : layout
Path-Layer: 3 Data Type: 0 No of points: 4 Width: 2000
Path Type: Truncate
(7500,0) (7500,7500) (14500,7500) (14500,2500)
Polygon - Layer : 2 Data Type : 0 No of points : 9
(4500,1000) (0,1000) (0,3000) (1500,3000) (1500,4500)
(3000,4500) (3000,3500) (4500,3500) (4500,1000)
Rectangle - Layer : 1 Data Type : 0 BBOX :
(10500,1500) (12500,3500)
End Cell Definition
End Library
Using Export – Stream

*Export – Stream* translates dfII database to GDSII Stream mask data format. In addition, you can use *Export – Stream* to check geometric data, remap layers, create a text version of Stream data, and extract symbolic data.

The following figure shows the files *Export – Stream* uses and generates.

The following list provides generic information about the files you provide to *Export – Stream*:

- **Design Framework II database**
  This is the Design Framework II database that you want to translate to a Stream file.

- **Template file**
  This file contains the parameter settings you want to use to run *Export – Stream*. You can load the parameter settings in the template file into the *Virtuoso® Stream Out* form. If you use the *pipo* command to translate your Design Framework II library file, you must specify a template file. (*The *pipo* command is the noninteractive, batch mode version of *Import – Stream* and *Export – Stream.*) You can create a template file by setting the values you want in the *Virtuoso® Stream Out* form and clicking the *Save* button in the form. Otherwise, you can copy the sample template file from `samples/transUI/streamout.il` and modify it.
Property map file
This is an optional file you can use to map object types and property names from the Design Framework II library file to the Stream file. For detailed information about creating a property map table, see the section “Property Mapping File for Stream Out” on page 42.

Cell name map file
This is an optional file you can use to map the Design Framework II cell names to the cell and view names you want to use in the Stream file. For detailed information about creating a cell name map table, see the section “Cell Name Mapping File for Stream Out” on page 37.

Text font map file
This is an optional file you can use to map the font names used in the Design Framework II database to the font numbers you want to use in the Stream file. For detailed information about creating a text font map table, see the section “Text Font Mapping File for Stream Out” on page 41.

Layer name map file
This is an optional file you can use to map the layer numbers used in the Design Framework II database to the layer names you want to use in the Stream file. For detailed information about creating a layer name mapping file, see the section “Layer Name Mapping File for Stream Out” on page 37.

Pin text mapping file
This is an optional file you can use to map pins in the Design Framework II database to labels in the Stream file. For detailed information about creating a pin text mapping file, see the section “Pin Text Mapping File for Stream Out” on page 44.

SKILL file
You can use a SKILL file to map cell names, font types, and layer names: and handle illegal shapes, such as open or reentrant polygons. For detailed information about using a SKILL file, see the section “Using SKILL to Customize Stream and CIF Files” on page 269.

Note: Export – Stream reads the SKILL file for a particular type of mapping under certain conditions. For more information about when Export – Stream uses the SKILL procedures, refer to the section “Using Map Files when Exporting” on page 36.

Export – Stream produces either a Stream file or an ASCII Dump file (which is a text interpretation of the Design Framework II database) and an error file.
Stream file

This is the output Stream file. Export – Stream writes information, warning, and error messages in the error file during translation so that you can check the error file for problems encountered, if any, during translation.

When Export - Stream is invoked from GUI, it will check if an output stream file exists and show a pop-up box asking if you intend to overwrite the file. If you select the yes option, it will overwrite the file. If you select the no option, then Export - Stream will stop there.

However, when Export - Stream is invoked directly in the batch mode, for example, by running the pipo command, then it will always overwrite any existing output Stream file.

ASCII dump

This is a text interpretation of how Export – Stream translates the devices in your Design Framework II library file. When you produce an ASCII dump file, Export – Stream does not translate any data.

Error file

This is the file that contains all the informational, warning, and error messages that Export – Stream generates during translation. You can set several parameters on the Stream Out Options form to filter the types of messages that Export – Stream writes to the error file.

pio_xout_info

This file contains information about the complete path, name, view, and the version of a cell. If the cell is a submaster or pcell variant, the additional information is also dumped as:

pcell variant: <Variant name>

pio_Pcell_info

This file consists information about the pcell name, pcell variant name, and the property value pairs. This file is created only when the pcell suffix is dbID or dbIDPlusTime. The user has enough information about the parameters of the variants. It can be selectively dumped using the Dump Pcell Info option.
Export – Stream Forms

File ➔ Export ➔ Stream...

Translates files from dfII database format to GDSII Stream format.

**Note:** Stream Out now translates cellviews in the bottom-up order. So, the leaf-level cells are translated first followed by the top-level cells.

This section describes the following Export - Stream forms:

- [Stream Out Form](#) on page 95
- [Stream Out User-Defined Data Form](#) on page 101
- [Stream Out Options Form](#) on page 106

For a list of equivalent template file options, refer to the Stream Out GUI and Template File Options section.
Stream Out Form

User-Defined Data And Options are buttons that bring up other forms.

**User-Defined Data**

The *User-Defined Data* button displays the “Stream Out User-Defined Data Form” on page 101. This form lets you specify optional pin conversion information, mapping tables, and user-defined data, for example, *User-Defined Property Mapping File*, *User-Defined Property Separator*, and *User-Defined SKILL File*.

**Options**

The *Options* button displays the “Stream Out Options Form” on page 106. This form contains more parameters that help you specify how Export – Stream translates your
Design Framework II database file. The Stream Out Options form also contains parameters that let you filter the messages that Export – Stream writes to the error file.

**Set Fast Options**

Use the Set Fast Options button to set certain performance intensive options available in Stream Out to their default values. These options are listed below:

- Convert Pcells to Geometry
- Hierarchy Depth Limit
- Maximum Vertices in Path/Polygon
- Convert Paths to Polygons
- Report Bad Polygons
- Convert Simple Mosaic to Array
- Precision Report
- Techfile Choice
- Dump Pcell Info
- Rod Directory
- Keep Pcells
- *Keep Pin Information as Attribute Number*

During Stream Out, PIPO performance might be negatively impacted with the values that you specify for these options. This happens because they require extra processing of data during translation. The negative impact on PIPO performance can be prevented by setting the default values of such options before exporting Stream files.

When you use Set Fast Options, a dialog box appears and displays a warning message about overwriting the values you might have already specified with the default values of the Stream Out options.

The Set Fast Options option does not update the template file unless you save it by using the Save option. For more information on PIPO performance, see Guidelines for Stream Out.

**Template File**

Use the Template File field to supply the filename and option settings for the Virtuoso® Stream Out form. When you use Export – Stream, a template file is optional. However, if you use the pipo command, you must create and specify a template file.
Load
The Load button updates the form with the option settings in the template file.

Save
The Save button writes the current option settings to the template file.

Browse
The Browse button opens a UNIX Browser window to let you search for the template file.

Library Browser
The Library Browser button displays the Library Browser tool. You can use the Library Browser to enter a library name, a top cell name, and a view name in the three fields below Run Directory.

Run Directory
Use the Run Directory field if you want to read from or write to a file that is not in the run directory. It is the default directory for all the files you specify in the Virtuoso® Stream Out form. Default: current directory (.)

Library Name
Use the Library Name field to translate the Design Framework II database file to Stream format.

Note: Stream Out maps the dfII database library name to an upper case name with a .DB extension in the GDS file. For example, if the name of the library is geomLIB, the corresponding library name embedded in the GDS file generated by Stream Out is GEOMLIB.DB.

Top Cell Name
The Top Cell Name field accepts the name of the cell at the top of the hierarchy that you want to translate, the name of the view that you want to translate, and the version number of the view. If you omit the view name, Export – Stream uses the view name specified in the View Name field. Export – Stream uses the version number for the top cellview only. All subsequent cellviews in the hierarchy become the latest version in the library. If you do not specify a version number for the top cell, Export – Stream uses the latest version.

You can enter multiple top cellviews. Use a semicolon (;) to separate each top cellview and a blank space to separate the cell name and the view name.

The syntax is:
You can enter as many top cells as you want. The default value is an empty field, which translates the entire library. For example, assume the cell hierarchy is as described below. The following table shows some examples of how you can translate cell hierarchies:

<table>
<thead>
<tr>
<th>Top Cells to Translate</th>
<th>View to Translate</th>
<th>Version to Translate</th>
<th>What to type in the field</th>
</tr>
</thead>
<tbody>
<tr>
<td>B D</td>
<td>layout*</td>
<td>latest</td>
<td>B;D</td>
</tr>
<tr>
<td>B C</td>
<td>layout2, layout*</td>
<td>1.2, latest</td>
<td>B layout2 1.2;C</td>
</tr>
<tr>
<td>C</td>
<td>layout*</td>
<td>2.1</td>
<td>C 2.1</td>
</tr>
<tr>
<td>All cells</td>
<td>layout*</td>
<td>latest</td>
<td></td>
</tr>
<tr>
<td>All cells</td>
<td>layout3</td>
<td>3.0</td>
<td>layout3 3.0*</td>
</tr>
</tbody>
</table>

* This example assumes that you use the default value layout in the View Name field.

After you enter the top cell name in the Top Cell Name option, the Output File option will reflect the filename as `<Top Cell Name>.<file extension>`. The default file extension name for the output file is .gds. You can change the default file extension name from .gds to any other string by using the streamFileExt SKILL variable. You can also change the Output File option to any other name. The limit for the cell name is 1024 characters during export.

View Name

Use the View Name field during Export – Stream only for the top cells for which you do not specify a view name. It is the default view name for the top cells specified in the Top Cell View option.

Default: layout

Output
The **Output** option selects the output data format that *Export – Stream* creates.

**Stream DB**

The *Stream DB* option translates the data of the Design Framework II database to a Stream file.

**ASCII Dump**

The *ASCII Dump* option translates the Design Framework II library to ASCII format. If you set this option on, *Export – Stream* does not translate the Design Framework II database. When you set this option on, you must specify a filename in the *ASCII Dump File* field.

**ASCII Dump File**

Use the *ASCII Dump File* option when *Export – Stream* writes a text interpretation of the Design Framework II database. You must type a filename when you set the **Output** option to *ASCII Dump*.

**Output File**

Use the *Output File* field when you want to translate a Stream file to the Design Framework II database. You must type an output file if you selected *Stream DB* as your *Output* option. Use the *Browse* button to search for the output file.

**Compression**

Use the *Compression* option to compress the output Stream file. Supported utilities for compressing files are **gzip**, **bzip2**, and **compress**.

**gzip**

Use the *gzip* utility to compress the output Stream file. If the output Stream file does not have `.gz` as the file extension, it adds `.gz` to the file name.

For example, the output Stream file `out.gds` changes to `out.gds.gz` but if the file name is `out.gds.gz`, then the file name remains unchanged. If *gzip* is not present in **PATH**, an uncompressed Stream file is generated and it has `.pipoTMP` as the file extension.

**bzip2**

Use the *bzip2* utility to compress the output Stream file. If the output Stream file does not have `.bz2` as the file extension, it adds `.bz2` to the file name.

For example, the output Stream file `out.gds` changes to `out.gds.bz2` but if the file name is `out.gds.bz2`, then the file name remains unchanged. If *bzip2* is not present in **PATH**, an uncompressed Stream file is generated and it has `.pipoTMP` as the file extension.
PATH, an uncompressed Stream file is generated and it has .pipoTMP as the file extension.

**compress**

Use the `compress` command to compress the output Stream file. If the output Stream file does not have .Z as the file extension, it adds .Z to the file name.

For example, the output Stream file `out.gds` changes to `out.gds.Z` but if the file name is `out.gds.Z`, then the file name remains unchanged. If `compress` is not present in PATH, an uncompressed Stream file is generated and it has .pipoTMP as the file extension.

**none**

Use the `none` option if you do not want to compress the output Stream file.

Default: `none`

**Scale UU/DBU**

It is the ratio of database units per user unit you want in the output Stream file. Use the `Scale UU/DBU` field to change the precision of your design, but not to scale your design. For information on scaling your design, refer to the section “Scaling Your Design” on page 24.

Default: 0.001

**Units**

Use the `Units` option to specify a basic unit of length in the output Stream file.

- **micron**
  
  The `micron` option sets the unit to 1 micron (0.001 millimeter).

- **millimeter**
  
  The `millimeter` option sets the unit to 1 millimeter.

- **mil**
  
  The `mil` option sets the unit to 1 mil (approximately 25.4 microns).

**Process Nice Value 0-20**

The `Process Nice Value 0-20` option lets you adjust the nice priority value. The nice priority value is a factor used by the UNIX operating system to schedule the CPU time allocated to a
process. The larger the value, the longer it takes Export – Stream to translate your design.

Valid Values: An integer between 0 and 20, with 0 being highest priority. Default: 0

Error Message File

Use the Error Message File field during Export – Stream to write informational, warning, and error messages. Use the Browse button to search for the error message file. Default: PIPO.LOG

Stream Out User-Defined Data Form

Convert Pin to

The Convert Pin to option translates pins into cell instances and labels or ignores them. If you select text, you must specify a pin text map table filename.

- geometry
  
  The geometry option translates pins into shapes or instances.

- text
  
  The text option translates pins into labels.

- geometry & text
The *geometry & text* option translates pins into shapes or instances and labels.

**drop**

The *drop* option does not translate pins.

**Note:** If `cdsVia` is used in the input design and during Stream Out, if *Text* is chosen as the value for the *Convert Pin to* option, then any geometry pin inside `cdsVia` will be translated both as geometry and text. Similarly, if the `cdsVia` instance is a pin, then it will be translated both as SREF and text.

**Pin Text Map Table**

The *Pin Text Map Table* field lets you specify a file containing mapping information between the types of pins in your design and labels. You must specify this file when you set *Convert Pin to text*. For more information about how to create a property mapping file, refer to the section “Pin Text Mapping File for Stream Out” on page 44.

**Note:** You must specify a file when you set *Convert Pin to text*. Even if all the pins in your design are geometry pins and your mapping file is empty, you must specify the file.

**Keep pin information as attribute number**

The *Keep pin information as attribute number* field indicates whether the system describes pin information (such as *pin name*, *access direction*, *terminal name*, and *direction*) as a Stream attribute. To use this option, you must set the *Convert Pin to* option to either *geometry* or *text*. If you specify *text*, you must supply an empty mapping filename in the *Pin Text Map Table* option. Valid Values: 0 or an integer from 1 to 127. Default: 0 (no pin information is shown)

The pin information is shown in the Stream file in the following sequence:

\[
\text{access\_dir\ } \text{pinName\ } \text{terminalName\ } \text{direction\ } \text{[instName]}
\]

- **access\_dir**
  The access direction of the pin, which can be T for top, B for bottom, L for left, and R for right. If the pin has more than one access direction, the directions appear in the sequence TBLR. If the direction is unknown, Export – Stream writes one blank character.

- **pinName**
  Name of the pin

- **terminalName**
  Name of the terminal

- **direction**
  Direction of the terminal. Valid Values: input, output, inputOutput, switch, jumper, unused, unknown
For example, if the pin name is 13, the terminal name is VSS, the access direction is Bottom, and the terminal direction is inputOutput, Export – Stream gives the attribute number you type in this field the value B 13 VSS inputOutput and assigns it to the instance or rectangle pin.

**Cell Name Map Table**

The *Cell Name Map Table* field lets you optionally specify a file for mapping cell names from the Design Framework II database to the output Stream files. You can also specify cell name mapping information in a SKILL file. You specify the SKILL file in the *User-Defined SKILL File* field. If you do not provide cell mapping information, Export – Stream uses the Case Sensitivity option setting from the Stream Out Options form.

For details about how Export – Stream translates cell names and detailed information about how to create a cell name map table, refer to the section “Cell Name Mapping File for Stream Out” on page 37.

**Note:** Export – Stream will automatically change cell names while writing the Stream file if it finds instances of two or more master cellviews with the same name but from different libraries and/or having different viewnames in the dfII database.

**Example**

Instances of *Lib1 cellA view1; Lib1 cellA view2*

Instances of *Lib1 cellA view1; Lib2 cellA view1*

During Export – Stream, a new cell name map table `cellName.map` is created. During Import – Stream, it can be included with the `cellMapTable` option in the template file or the *Cell Name Map Table* option in the Stream In User-Defined Data form to preserve cell name mapping. You might also need to append additional cell mapping information to this file if you specify a custom cell name map table during Export – Stream. For details on how to modify a cell name map file for import – Stream, see Chapter 2, “Using Map Files When Importing.”.

⚠️ **Caution**

You might have to modify the `cellName.map` file before running Stream In because of the inherent differences in the way the layout data is organized in dfII and the Stream format.

**Layer Map Table**
The Layer Map Table field lets you optionally specify a file for mapping Design Framework II database layer-purpose pairs to Stream layers. This file is used during Stream Out. This is useful for changing layers, merging different layers into one layer, and filtering layers. You can also specify layer mapping information in a SKILL file. You specify the SKILL file in the User-Defined SKILL File field. If you specify a layer map table, Export – Stream does not check a SKILL file for layer mapping.

If you do not specify a layer map table or a SKILL file that contains layer mapping routines, only layers with the streamTranslateLayer property set to t are translated. The streamLayerNumber property defines the Stream layer number to use. The streamDatatypeNumber property defines the Stream datatype number to use. You can edit these properties in the database you want to translate.

For detailed information about how to create a layer map table, refer to the section “Layer Name Mapping File for Stream Out” on page 37.

Text Font Map Table

The Text Font Map Table field lets you optionally specify a file for mapping Design Framework II text fonts and font sizes to Stream text fonts and font sizes. You can also specify text font mapping information in a SKILL file. You can also specify the SKILL file in the User-Defined SKILL File field. If you specify a text font map table, Export – Stream does not check a SKILL file for font mapping.

For detailed information about how to create a text font map table, refer to the section “Text Font Mapping File for Stream Out” on page 41.

User-Defined Property Mapping File

The User-Defined Property Mapping File field lets you optionally specify a file for mapping object types and property names from the Design Framework II database to the Stream file. If you do not specify a filename, Export – Stream does not translate any properties. You can specify the character Export – Stream uses to separate the property string with the User-Defined Property Separator option.

For more information about how to create a User-Defined Property Mapping file, refer to the section “Property Mapping File for Stream Out” on page 42.

User-Defined Property Separator

The User-Defined Property Separator field lets you specify a character to separate properties in a Stream file. This character is used during Stream Out. Most Stream files use a comma to separate property values, but some files use other characters or spaces. If you specify a SEPARATOR in a User-Defined Property Mapping File, Import-Stream ignores the Property Separator defined in the form. Default: a comma (,)
User-Defined SKILL File

The *User-Defined SKILL File* field lets you specify a file that contains user-defined SKILL routines. These SKILL routines are used by the system to map cell names, parameterized cell names, layer names, and text fonts, and to translate illegal shapes. For information about writing SKILL routines, see the section “Customizing Export Files” on page 279 or refer to the *SKILL Language User Guide* and *SKILL Language Reference*. 
Stream Out Options Form

<table>
<thead>
<tr>
<th>Stream Out Options</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>Cancel</td>
</tr>
<tr>
<td>Default</td>
<td>Apply</td>
</tr>
<tr>
<td>Help</td>
<td></td>
</tr>
<tr>
<td>Report Bad Polygons</td>
<td></td>
</tr>
<tr>
<td>To Use Parent XY for Text</td>
<td></td>
</tr>
<tr>
<td>Output Rectangle as BOX</td>
<td></td>
</tr>
<tr>
<td>Snap XY to Grid Resolution</td>
<td></td>
</tr>
<tr>
<td>Convert Simple Mosaic to Array</td>
<td>■</td>
</tr>
<tr>
<td>Convert PCells to Geometry</td>
<td></td>
</tr>
<tr>
<td>Convert Lines to</td>
<td></td>
</tr>
<tr>
<td>Case Sensitivity</td>
<td></td>
</tr>
<tr>
<td>Text Case Sensitivity</td>
<td></td>
</tr>
<tr>
<td>Convert Dots to</td>
<td></td>
</tr>
<tr>
<td>Keep PCells</td>
<td></td>
</tr>
<tr>
<td>Replace &lt;&gt; with []</td>
<td></td>
</tr>
<tr>
<td>Convert Paths to Polygons</td>
<td></td>
</tr>
<tr>
<td>Library Version</td>
<td>5.0</td>
</tr>
<tr>
<td>Precision Report</td>
<td></td>
</tr>
<tr>
<td>Retain Reference Library (No Merge)</td>
<td></td>
</tr>
<tr>
<td>Filter Out Warning/Information Messages</td>
<td></td>
</tr>
<tr>
<td>Filter Out Unmapping Warning</td>
<td></td>
</tr>
<tr>
<td>Techfile Choice</td>
<td></td>
</tr>
<tr>
<td>PCell Suffix</td>
<td></td>
</tr>
<tr>
<td>Hierarchy Depth Limit</td>
<td>3 4</td>
</tr>
<tr>
<td>Maximum Vertices in Path/Polygon</td>
<td>20 9</td>
</tr>
<tr>
<td>Rod Directory</td>
<td></td>
</tr>
<tr>
<td>Respect GDSII limits</td>
<td></td>
</tr>
<tr>
<td>Dump PCell Info</td>
<td></td>
</tr>
<tr>
<td>Comprehensive Log</td>
<td></td>
</tr>
<tr>
<td>Reference Library List</td>
<td></td>
</tr>
<tr>
<td>Ignore PCell evaluation failure</td>
<td></td>
</tr>
<tr>
<td>Generate Hierarchy Listing</td>
<td></td>
</tr>
</tbody>
</table>
Report Bad Polygons

The Report Bad Polygons option determines whether Export – Stream removes collinear and coincident points in polygons and paths, and reports incomplete or illegal polygons in the error file. A polygon can have more than one error, although Export – Stream reports only the most severe problem in the error file. Export – Stream reports the following problems, listed here in order of severity with the most severe problem described first.

- **Open**: A polygon whose first and last coordinates do not coincide.
- **Reentrant**: A polygon in which the sum of the internal angles is not 360 degrees or in which one or more pairs of edges intersect rather than meet.
- **Acute angle**: An angle that is smaller than 90 degrees.
- **Non-45 degree angle**: An angle that is not a multiple of 45 degrees, for example 60.

Valid Values: 90, 135, 180, 225, 270, 315, 360

**Note**: An angle of a polygon refers to the internal angle of the polygon.

To Use Parent XY for Text

The To Use Parent XY for Text option controls the position of text. When turned on, it uses as its origin the point at which the text attaches to the shape. Default: off

Output Rectangle as BOX

The Output Rectangle as BOX option indicates whether you want to translate rectangles into polygons or BOX records in Stream format. A BOX record is an alternative way of defining a rectangular shape. The record consists of a coordinate for the center of the shape and values for the height and width of the shape. When you turn this option on, Export – Stream translates rectangles to BOX records. When you turn this option off, Export – Stream translates rectangles to polygons. Default: off

Snap XY to Grid Resolution

The Snap XY to Grid Resolution option adjusts the coordinates of geometric objects to coincide with the grid resolution. This lets graphic commands (such as the Virtuoso Layout Editor commands Move, Stretch, and Split) access end points, edges, and vertexes of geometric objects. However, this option can shift data from the original Stream grid, resulting in a loss of precision. Default: off
Note: You can set the grid resolution using mfgGridResolution in the Physical Rules class of the technology file you specify in the ASCII Technology File Name field on the “Stream In Form” on page 57. If you do not define mfgGridResolution, Import – Stream sets it to the inverse of the value you set for the Scale UU/DBU field on the “Stream In Form” on page 57.

Convert Simple Mosaic to Array

The Convert Simple Mosaic to Array option converts each Design Framework II mosaic to a Stream array. In other words, if you select this option, Export – Stream will translate the mosaic to an AREF. You can Stream In the AREF back to a single Design Framework II mosaic.

Note: The term ‘Array’ in the option name refers to the AREF of Stream format.

However, if you turn off this option, Export – Stream will translate each element of the mosaic to an SREF. In this case, you cannot Stream In the individual cells to obtain the original Design Framework II mosaic.

Default: on.

Convert PCells to Geometry

The Convert PCells to Geometry option flattens parameterized cell instances in the design to create polygon layout cellviews.

Default: off.

Do not preserve pins

The Do not preserve pins sub-option does not retain the connectivity information of pins in flattened pcell instances. This option is visible only if you select the Convert PCells to Geometry option.

Default: off.

Caution

The Convert PCells to Geometry option cannot be used with the Keep PCells option simultaneously in a single run of Stream Out. This is because the Keep PCells option is used to preserve parameterized cells during the round-trip of Stream Out and Stream In which makes both these options mutually exclusive to each other in functionality. Therefore,
if you use both these options simultaneously in single run of Stream Out, Export - Stream will automatically turn off the Convert PCells to Geometry option, and issue a warning message.

Note: During Export – Stream, when the Convert PCells to Geometry option is used, symbolic pins are converted to dots. Therefore, the Convert Pin to option is ignored for these symbolic pins.

Convert Lines to

The Convert Lines to option determines whether Export – Stream converts lines in the Design Framework II database to zero-width paths in the Stream file.

- **path**
  The path option translates lines to zero-width paths.

- **ignore**
  The ignore option does not translate lines.

Case Sensitivity

The Case Sensitivity option specifies whether Export – Stream changes the case of letters in cell names. Export – Stream uses this option setting when you do not supply cell name mapping information for the cell. You can map cell names in a cell name map table or in a SKILL file.

- **preserve**
  The preserve option translates cell names without changing case. For example, the name Obj1 remains Obj1.

- **upper**
  The upper option translates all cell names to uppercase. For example, the name Obj1 becomes OBJ1.

- **lower**
  The lower option translates all cell names to lowercase. For example, the name Obj1 becomes obj1.

Text Case Sensitivity

The Text Case Sensitivity option specifies whether Export – Stream changes the case of letters in text labels. Export – Stream uses this option when you do not supply text label
mapping information for the cell. You can map text labels in a SKILL file by using the `poTextMap()` function.

**Note:** If you specify the mapping information using the `poTextMap` function, the information specified in the *Text Case Sensitivity* option is ignored.

- **preserve**
  The `preserve` option translates text labels without changing case. For example, the label `Obj1` remains `Obj1`.

- **upper**
  The `upper` option translates the text label to uppercase. For example, the label `Obj1` becomes `OBJ1`.

- **lower**
  The `lower` option translates the text labels to lowercase. For example, the label `Obj1` becomes `obj1`.

### Convert Dots to

The *Convert Dots to* option determines whether `Export – Stream` converts dots in CDBA to polygons, nodes, or neither of these.

- **polygon**
  The `polygon` option translates dots to polygons.

- **node**
  The `node` option translates dots to nodes. The nodes are created at the same coordinates as the dots. Any properties on the dots are translated to node properties if the property names and numbers are specified in the property mapping file. The object type for the properties in the property mapping file should be `dot`. The order of the nodes written in the Stream file is sorted by the property numbers of the node properties.

- **ignore**
  The `ignore` option does not translate dots to polygons or nodes and ignores them.

**Caution**

*While converting dots to polygons, dots of zero size will be dropped.*

### Keep PCells
The \textit{Keep PCells} option preserves parameterized cells by dumping the parameterized information in separate SKILL files in the \texttt{KPDIR} directory, located in the run directory. Default: \texttt{off}

Stream format does not have direct support (direct mapping from CDBA to STREAM) for parameterized cells. Therefore, by default, the parameterized cells (cellview masters and their references) get translated to Stream as separate variants, also referred to as submasters. The parameterization information is therefore lost during one cycle of Stream Out and Stream In. The Keep PCell option provides a mechanism to preserve PCells during the cycle of Stream Out and Stream In. Refer to the \textit{Virtuoso Parameterized Cell Reference} for details on PCell variants.

The actual stream file generated by Stream Out is the same regardless of the setting of the \textit{Keep PCells} option. The intent here is to optionally store enough information externally so that when the Stream file is read back into dfII, the pcells can optionally be restored.

\textbf{Replace <> with []}

The \textit{Replace <> with [] option} replaces the character < with [ and > with ] in labels/textDisplays during the translation from the dfII library to a Stream file.

Default: \texttt{off}

\textbf{Convert Paths to Polygons}

The \textit{Convert Paths to Polygons} option converts each Design Framework II path to a polygon (boundary record) in Stream. Default: \texttt{off}

\textbf{Library Version}

The \textit{Library Version} option provides the version number of the output GDSII Stream. Default: \texttt{5.0}

\textbf{Note:} Stream versions 3.0 and earlier do not support the Stream path type 4 or the BOX stream records. If the version is set to 3.0 or lower, any paths of Stream path type 4 are converted to path type 0, and rectangles are not translated as box records even if the \textit{Output Rectangle as Box} option is set to \texttt{true}.

\textbf{Precision Report}

The \textit{Precision Report option} indicates whether \textit{Export – Stream} checks precision and writes a warning message to the error file if precision is lost during translation. A loss of precision occurs when \textit{Export – Stream} rounds a coordinate value and it falls off the grid. If you turn this option on, \textit{Export – Stream} checks precision and writes warnings to the error
file. When you turn this option on, Export – Stream is slowed down slightly. If you turn this option off, Export – Stream does not check precision. Default: off

**Retain Reference Library (No Merge)**

The Retain Reference Library (No Merge) option indicates whether you want to translate only the cell name and placement information for the reference library cells (which are not in the Design Framework II database you want to translate). For example, the source library (srcLib) contains a cell hierarchy cell A, which contains instances of cell B and cell C. Cells A and B exist in srcLib and cell C exists in the reference library refLib.

![Diagram of cell hierarchy]

If you turn this option off, Export – Stream merges the complete data definitions of cells A, B, and C from srcLib and refLib into the output Stream file. If you turn this option on, Export – Stream translates the complete definitions of cells A and B to the output Stream file but translates only the cell name placement information for cell C. For more information about translating reference libraries, refer to the section “Translating Reference Libraries Using Stream” on page 48.

**Note:** When the Retain Reference Library (No Merge) option is on, the Stream file generated has only the cell name and placement information, and the reference library relationship needs to be maintained during Stream In. The reference libraries should exist in cds.lib during Stream In. Ideally, you should keep this option on in both Stream Out and Stream In if you require the Stream creation and the library generation process to take place quickly and the complete information is not needed (consider an application such as the translation of abstracts). This option can be kept off in both Stream Out and Stream In if complete information is required in a Stream file (for running LVS on complete design). You might want to more permutations after referring to the details in the section “Translating Reference Libraries Using Stream” on page 48.

**Filter Out Warning/Information Messages**

The Filter Out Warning/Information Messages option helps you specify whether Export – Stream writes warning and information messages to the error file during translation. If you set this option on, Export – Stream writes only error messages to the error file. If you set this option off, Export – Stream writes all messages to the error file.

Default: off

**Filter Out Unmapping Warning**
The Filter Out Unmapping Warning option lets you choose whether Export – Stream writes warning messages about dfII layers to the error file during translation. Warnings about user layers are not affected.

Default: off

If you select this option, the following warnings will be filtered:

- **Message ID: 267**
  The Cadence design contains layer `layerNumber` which is not defined in the technology file. This layer will be ignored.

- **Message ID: 268**
  layer-purpose pair `layerName: purposeName` is not defined in the layer map file. This layer-purpose pair will be ignored.

- **Message ID: 335**
  Stream layer rule for the layer-purpose pair `layerName: purposeName` is nil. All objects on this LPP will be dropped.

- **Message ID: 347**
  Failed to get the Stream layer rules for the layer-purpose pair `layerNumber: purposeNumber`. All objects on this LPP will be dropped.

- **Message ID: 446**
  No mapping found for layer-purpose pair `layerName: purposeName`. All objects on this LPP will be dropped.

**Techfile Choice**

The Techfile Choice option lets you choose the technology file attached either to the input library or to each cellview traversed. The Design Framework II product can have any cellview attached to any technology file.

- **Input Library**
  The Input Library option selects the technology file attached to the library specified as an input to StreamOut.

- **Individual Cellview**
  The Individual Cellview option selects the technology file attached to each cellview traversed.
Pcell Suffix

The *Pcell Suffix* option selects what is appended to pcell names to make them unique in the stream file when pcells are translated.

**Database ID**

The *Database ID* option appends database IDs to pcell names. The translated names use the following format:

```
name$$<dbID>
```

The delimiter $$ can be customized with a user-defined SKILL function. This option is selected by default. For details, see *Chapter 6, Design Data Translators Reference*.

**Caution**

*Use of the Database ID option may lead to the creation of non-unique masters when similar pcells are streamed out in different sessions of PIPO. Therefore, this will cause one pcell to overwrite another when both are streamed into the same library. Use of this option for similar Pcells in a single PIPO session will create unique pcell submasters.*

**DbIdPlusTime**

The *DbIdPlusTime* option appends the database ID, along with date and time, to pcell names. The translated name is in the following format:

```
name$$<dbID><Date & Time>
```

The delimiter $$ can be customized with a user-defined SKILL function. For details, see *Chapter 6, Design Data Translators Reference*.

**Caution**

*The DbIdPlusTime option might lead to an increase in the size of the Stream file.*

This can be illustrated by the following example:

Consider a pcell, \( P \) with parameters \( W \) and \( L \) where,

- cell1 has the following two instances of pcell \( P \)
  - \( I1 : W = 4, L = 3 \)
  - \( I2 : W = 3, L = 4 \)
- and cell2 has the following instance of pcell \( P \)
  - \( I4 : W = 4, L = 3 \)

Instances \( I1 \) and \( I4 \) must have the same submaster and the same database ID. However, the processing time or the time for forming a variant can be different. This will lead to two
different submasters being created for I1 and I4. Therefore, although the data is absolutely correct and integrity is also maintained across different CDBA/PIPO sessions, it might lead to an increase in the Stream file size.

**Parameter Values**

The *Parameter Values* option appends parameter values to pcell names. This option is only for backward compatibility.

**Note:** Use of the *Parameter Values* option may create non-unique submasters in some situations. For example, the submasters of two supermaster pcells, `cell` with two parameters `x` and `y` and `cell_1` with one parameter `x`, are created as follows:

Submasters for `cell`:
- `cell_1_1` (param `x`:1, param `y`:1)
- `cell_1_2` (param `x`:2, param `y`:1)

Submasters for `cell_1`:
- `cell_1_1` (param `x`:1)
- `cell_1_2` (param `x`:2)

Note that the submasters for both the supermasters in this example have the same names.

**Caution**

*If Parameter Values is selected, *pip* might stop when translating very complex pcells. Complex Pcells are Pcells with a large number of parameters. The limit for the length of pcell names is 64 characters. If the length of a pcell name increases beyond 64, no further appends are made. If the last parameter that is appended increases the length beyond 64, pcell name will not be truncated to 64 and the name will be left as it is.*

**Hierarchy Depth Limit**

Use the *Hierarchy Depth Limit* field to specify the depth of the design hierarchy to be translated during Stream Out. Stream Out stops the translation process at the hierarchy level indicated by this option. Default: 32

**Note:** If the hierarchy depth of the input design is more than the value of the *Hierarchy Depth Limit* option, then one time warning message is displayed.

**Maximum Vertices in Path/Polygon**

The *Maximum Vertices in Path/Polygon* field indicates the maximum number of Stream vertexes you want to allow in a polygon. *Export – Stream* cuts any polygon or path that
contains more than this number of vertexes into smaller ones. Export – Stream sends a warning message to the error file for each cut polygon and path. Default: 200

**Note:** Polygons and paths are stored differently in Design Framework II and Stream formats. The dfII database does not store either the starting and ending points of a polygon because they are the same point. The Stream format requires that both points be stored individually. The largest Design Framework II polygon or path Export – Stream can process has 199 vertexes, which translates into 200 Stream vertexes.

Although the default is set to 200, which is the GDSII limit for vertexes, the maximum limit for vertexes in dfII is 4000. The maximum limit imposed by the GDSII format is 200 and that by the Stream format is 8000. The Stream/GDSII reader writer of each EDA vendor, is therefore, responsible for handling restrictions imposed by their databases with this option.

**Caution**

This option is intended for use with designs containing polygons with large numbers of vertexes, for example, 4000 or more. If you set this option to a value that causes Export - Stream to divide Design Framework II polygons into many Stream polygons, performance is reduced considerably and in some cases the translation fails.

**Rod Directory**

The **Rod Directory** field specifies the location of the ROD Directory for the PIPO-ROD interface to work. Stream Out creates ROD object geometric data in the Stream file from the dfII data. Stream Out stores ROD object name data on Property 126 with each Stream object that contains ROD data.

Stream Out creates one or more of the following auxiliary files in the ROD directory:

- **cellview_name.map** - A name-mapping file for each cellview. It maps the object name to Property 126 stored in the stream file.
- **cellview_name.il** - A SKILL file for each cellview. It contains ROD objects, with ROD constructs for each cellview.
- **constructFile** - A single library-level construct SKILL file containing the ROD alignments for all cellviews that contain ROD alignments.

**Respect GDSII limits**

The **Respect GDSII limits** option helps you specify respect/ignore the GDSII limits of 32 characters for cell name and library name and 64 numbers (0-63) for numbering stream layers.
and stream datatype. By default, the GDSII restriction is ignored. The length of cell name and library name will be truncated after 32 characters. In case of cell name, the modified name will appear in the cellName.map file, which is created for the cell name modifications. Entries in the layerMap file with stream layer/datatype number greater than 63 will be ignored when GDSII restriction is enabled.

Default: off

⚠️ Caution

*This option is only with regard to Stream Layer/DataType Number, Cell Name, and Lib Name GDSII limits.*

### Dump Pcell Info

The *Dump Pcell Info* option provides a choice to dump the pipo_pcell_info file conditionally. PIPO will dump the parameter information in pipo_pcell_info only if this option is `true`. The default behavior is kept as `false` because this will give improved performance to the user.

### Comprehensive Log

Select the *Comprehensive Log* option to output the warnings and error messages generated by SKILL function calls and PIPO dependencies into the PIPO log file.

Default: `nil`

### Reference Library List

Use the *Reference Library List* (refLibList) option to specify a list of reference libraries. These libraries must be separated by a space. The *Reference Library List* option works only when the *Retain Reference Library (No Merge)* (reflib) is set to `t`.

If the value of `reflib` is `nil`, then all the cells are translated in the Stream file. If the value of `reflib` is set to `t` and that of `refLibList` to "", then all the cells in the reference libraries are not translated to the Stream file. If the value of `reflib` is set to `t` and `refLibList` contains a set of reference libraries, then all the cells defined in those libraries are not translated to the Stream file. Only those cells, which are not defined in the reference libraries, are translated.

### Ignore Pcell evaluation failure

Select the *Ignore Pcell Evaluation Failure* option if you do not want PIPO to halt upon any pcell evaluation failure. If you do not select this option, PIPO will halt upon any pcell evaluation failure and a fatal message will be displayed.
Default: nil

**Generate Hierarchy Listing**

The *Generate Hierarchy Listing* option writes the hierarchical information in the log file. This is a boolean option. For example, when this option is selected, following information related to the top cells and hierarchy will be written to the PIPO log file:

```
Top Cells in File
----------------------
TOP   layout

List Hierarchy(#cellInst,#arrayInst)
-----------------------------------
TOP   layout
  .CELL4  layout(1)
    ..CELL3  layout(3)
      ...CELL2  layout(10)
        ....CELL1  layout(5)
```

Default: nil

While translating cdsVia or symbolic contact devices, PIPO streamOut writes BOUNDARY record for intermediate cut layer repeatedly based on the ‘Row’ and ‘Column’ values with which the device is instantiated. For example, if a symContact instance has M rows and N columns for the cut layer, the corresponding GDS will have MxN BOUNDARY records representing this cut layer. For large values of MxN, the size of the GDS file can grow very large due to this reason.

The `contactCutRowThresold` and `contactCutColThresold` options provide the flexibility of representing the cut layer geometry through AREF record in the GDS file.

By default PIPO doesn’t array the cut layer geometry. When used with non default values, if a cdsVia/ symContact instance has Row >= contactCutRowThresold and Column >= contactCutColThresold, PIPO creates in-memory cellview and an additional level of hierarchy. In addition, PIPO writes the regularly sized and spaced cut layers through AREF record in the GDS file and in turn reduces the GDS file size.

PIPO creates two in-memory cellviews - one cellview having only the cut-layer geometry. The second cellview has two layers - top and bottom - and has the first cellview instantiated as mosaic. These cellviews are written to the stream file instead of the original master of cdsvia/ SyContact device.

**Note:** Keep the following in mind while using the `contactCutRowThresold` and `contactCutColThresold` options:

- These options work only when translateTopDown option set to ‘t’ and keepPcell is set to ‘nil’. 
There are no GUI equivalent options for these.

Stacking for cdsVia devices is not supported through these options.

There is no support for both cdsVia and syContact devices with implant layers.

Preservation of devices on round-trip is not supported.

For example, let's consider a case where a `symContact` device is instantiated with three Rows and three Columns. By default PIPO will generate following GDS file.

```
HEADER 5
BGNLIB LASTMODTIME LASTXSTIME
LIBNAME=NEWDESIGN.DB
UNITS 1.000000e-03 1.000000e-09
BGNSTR LASTMODTIME LASTXSTIME
STRNAME testcase5
SREF SNAME=via1$$34488364 STRANS=0 X=1405 Y=2500 ENDEL
ENDSTR
BGNSTR LASTMODTIME LASTXSTIME
STRNAME via1$$34488364
BOUNDARY LAYER=2 DATATYPE=0 XY=(-660,-670) (710,-670) (710,660) (-660,660) (-660,-670) ENDEL
BOUNDARY LAYER=0 DATATYPE=0 XY=(-710,-660) (660,-660) (660,670) (-710,670) (-710,-660) ENDEL
BOUNDARY LAYER=1 DATATYPE=0 XY=(390,390) (650,390) (650,650) (390,650) (390,390) ENDEL
BOUNDARY LAYER=1 DATATYPE=0 XY=(130,390) (390,390) (390,650) (130,650) (130,390) ENDEL
BOUNDARY LAYER=1 DATATYPE=0 XY=(-130,390) (130,390) (130,650) (-130,650) (-130,390) ENDEL
BOUNDARY LAYER=1 DATATYPE=0 XY=(-390,390) (-130,390) (-130,650) (-390,650) (-390,390) ENDEL
BOUNDARY LAYER=1 DATATYPE=0 XY=(-650,390) (-390,390) (-390,650) (-650,650) (-650,390) ENDEL
BOUNDARY LAYER=1 DATATYPE=0 XY=(390,130) (650,130) (650,390) (390,390) (390,130) ENDEL
BOUNDARY LAYER=1 DATATYPE=0 XY=(130,130) (390,130) (390,390) (130,390) (130,130) ENDEL
BOUNDARY LAYER=1 DATATYPE=0 XY=(-130,130) (130,130) (130,390) (-130,390) (-130,130) ENDEL
BOUNDARY LAYER=1 DATATYPE=0 XY=(-390,130) (-130,130) (-130,390) (-390,390) (-390,130) ENDEL
ENDSTR
ENDLIB
```

---
When streamed out with contactCutRowThreshold and contactCutColThreshold with value three, following stream file will be generated.

```
HEADER 5
BGNLIB LASTMODTIME LASTXSTIME
LIBNAME=NEWDESIGN.DB
UNITS 1.000000e-03 1.000000e-09
BGNSTR LASTMODTIME LASTXSTIME
STRNAME testcase5
SREF SNAME=Metal1_120204020_Metal2_201202040_Via12520520_3_3_0_0  STRANS=0
X=1405  Y=2500  ENDEL
ENDSTR
BGNSTR LASTMODTIME LASTXSTIME
STRNAME Metal1_120204020_Metal2_201202040_Via12520520_3_3_0_0
BOUNDARY  LAYER=2  DATATYPE=0  XY=(-660,-670) (710,-670) (710,660) (-660,660) (-660,-670) ENDEL
BOUNDARY  LAYER=0  DATATYPE=0  XY=(-710,-660) (660,-660) (660,670) (-710,670) (-710,-660) ENDEL
AREF SNAME=Via12_520_520  STRANS=0  COLUMNS=5  ROWS=5  XY=(-650,-650) (650,-650) (-650,650) ENDEL
ENDSTR
BGNSTR LASTMODTIME LASTXSTIME
STRNAME Via12_520_520
BOUNDARY  LAYER=1  DATATYPE=0  XY=(0,0) (260,0) (260,260) (0,260) (0,0) ENDEL
ENDSTR
ENDLIB
```

**Starting Stream Out from a Form**

1. Select *File – Export – Stream* from the CIW.
   
   The *Virtuoso* Stream Out form appears.

2. Type the name of the Design Framework II library to translate.

3. Type the output filename.

4. Specify any optional fields you want.

5. Click *Apply or OK*.

   *Export – Stream* begins the conversion. Check the CIW for the job number and information about translation success or failure.
At the end of execution, a pop-up box will show the status of your runs. Here is an example of such a pop-up box

![STRMOUT PopUp Message]

PIPO STRMOUT (PID = ipc:0) completed successfully!!

*** There were no error or warning message ***

This pop-up box contains the job number of your run. For example, (PID = ipc:0). It also gives you the status of your run, along with the number of errors or warnings if any.

The Display Log button displays the contents of the log file.

Starting Export – Stream from the Command Line

1. Create a template file by using the *Virtuoso® Stream Out* form.

2. For information about how to create the template file, see the section “Preparing a Template File and a .cdsinit File for Stream Out” on page 124.

3. Type the following at the command line:

   pipo strmout templateFile
Stream Out GUI and Template File Options

The following table provides a list of all the Stream Out GUI options and their corresponding names in the template file.

<table>
<thead>
<tr>
<th>Stream Out GUI</th>
<th>Stream Out Template File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run Directory</td>
<td>runDir</td>
</tr>
<tr>
<td>Library Name</td>
<td>libName</td>
</tr>
<tr>
<td>Top Cell Name</td>
<td>primaryCell</td>
</tr>
<tr>
<td>View Name</td>
<td>viewName</td>
</tr>
<tr>
<td>Output File</td>
<td>outFile</td>
</tr>
<tr>
<td>Scale UU/DBU</td>
<td>scale</td>
</tr>
<tr>
<td>Units</td>
<td>units</td>
</tr>
<tr>
<td>Compression</td>
<td>compression</td>
</tr>
<tr>
<td>Hierarchy Depth Limit</td>
<td>hierDepth</td>
</tr>
<tr>
<td>Convert PCells to Geometry</td>
<td>convertToGeo</td>
</tr>
<tr>
<td>Maximum Vertices in Path/Polygon</td>
<td>maxVertices</td>
</tr>
<tr>
<td>Retain Reference Library (No Merge)</td>
<td>refLib</td>
</tr>
<tr>
<td>Library Version</td>
<td>libVersion</td>
</tr>
<tr>
<td>Report Bad Polygons</td>
<td>checkPolygon</td>
</tr>
<tr>
<td>Snap XY to Grid Resolution</td>
<td>snapToGrid</td>
</tr>
<tr>
<td>Convert Simple Mosaic to Array</td>
<td>simMosaicToArray</td>
</tr>
<tr>
<td>Case Sensitivity</td>
<td>caseSensitivity</td>
</tr>
<tr>
<td>Convert Lines to</td>
<td>lineToZeroPath</td>
</tr>
<tr>
<td>Convert Dots to</td>
<td>convertDot</td>
</tr>
<tr>
<td>Output Rectangle as BOX</td>
<td>rectToBox</td>
</tr>
<tr>
<td>Convert Paths to Polygons</td>
<td>convertPathToPoly</td>
</tr>
<tr>
<td>Keep PCells</td>
<td>keepPcell</td>
</tr>
<tr>
<td>Replace &lt;&gt; with []</td>
<td>replaceBusBitChar</td>
</tr>
<tr>
<td>To Use Parent XY for Text</td>
<td>useParentXYforText</td>
</tr>
</tbody>
</table>
## Stream Out GUI

<table>
<thead>
<tr>
<th>Stream Out GUI</th>
<th>Stream Out Template File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision Report</td>
<td>reportPrecision</td>
</tr>
<tr>
<td>Filter Out Warning/Information Messages</td>
<td>runQuiet</td>
</tr>
<tr>
<td>Comprehensive Log</td>
<td>comprehensiveLog</td>
</tr>
<tr>
<td>Ignore Pcell evaluation failure</td>
<td>ignorePcellEvalFail</td>
</tr>
<tr>
<td>Error Message File</td>
<td>errFile</td>
</tr>
<tr>
<td>Filter Out Unmapping Warning</td>
<td>NOUnmappingLayerWarning</td>
</tr>
<tr>
<td>Techfile Choice</td>
<td>techFileChoice</td>
</tr>
<tr>
<td>Pcell Suffix</td>
<td>pcellSuffix</td>
</tr>
<tr>
<td>Respect GDSII limits</td>
<td>respectGDSIILimits</td>
</tr>
<tr>
<td>Generate Hierarchy Listing</td>
<td>genListHier</td>
</tr>
<tr>
<td>Dump Pcell Info</td>
<td>dumpPcellInfo</td>
</tr>
<tr>
<td>Cell Name Map Table</td>
<td>cellMapTable</td>
</tr>
<tr>
<td>Layer Map Table</td>
<td>layerTable</td>
</tr>
<tr>
<td>Text Font Map Table</td>
<td>textFontTable</td>
</tr>
<tr>
<td>Convert Pin to</td>
<td>convertPin</td>
</tr>
<tr>
<td>Keep pin information as attribute number</td>
<td>pinInfo</td>
</tr>
<tr>
<td>Pin Text Map Table</td>
<td>pinTextMapTable</td>
</tr>
<tr>
<td>User-Defined Property Mapping File</td>
<td>propMapTable</td>
</tr>
<tr>
<td>User-Defined Property Separator</td>
<td>propSeparator</td>
</tr>
<tr>
<td>User-Defined SKILL File</td>
<td>userSkillFile</td>
</tr>
<tr>
<td>Rod Directory</td>
<td>rodDir</td>
</tr>
<tr>
<td>Reference Library List</td>
<td>refLibList</td>
</tr>
<tr>
<td>NA</td>
<td>translateTopDown (See the note below.)</td>
</tr>
</tbody>
</table>

### Important

The `translateTopDown` option in the Stream Out template file does not have a corresponding GUI option.
By default, cells in the hierarchy are translated bottom to up. To specify the translation to happen in the top down manner, set the `translateTopDown` option in the template file to `t`. The table below summarizes the value settings for the `translateTopDown` option.

<table>
<thead>
<tr>
<th>Values of the <code>translateTopDown</code> Option</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>nil</code> or option not defined in the template file</td>
<td>Bottom down translation</td>
</tr>
<tr>
<td><code>t</code></td>
<td>Top down translation</td>
</tr>
</tbody>
</table>

### Preparing a Template File and a .cdsinit File for Stream Out

A collection of option values that you can load into the *Virtuoso® Stream Out* form can be defined in a template file or in the `.cdsinit` file. The required values in the Stream Out template file and the `.cdsinit` file are exactly the same. You can use either file to load the option values into the *Virtuoso® Stream Out* form.

You can load a template file into the *Virtuoso® Stream Out* form in the interactive mode or when you use the `pipocmd` command. The `.cdsinit` file is automatically loaded into the *Virtuoso® Stream Out* form when you specify `layout` as your view type. You can create a template file in one of two ways:

- Enter values in the *Virtuoso® Stream Out*, *Stream Out Options*, and *Stream Out User-Defined Data* forms, and click `Save` to save the option values to the file you specify in the `Template File` field.

- Create a template file with a text editor. You can copy the sample Stream Out template file called `samples/transUI/streamOut.il` and modify it. The only required values in the Stream Out template file are the names of the input dfII database and the Stream file. The remaining values are optional. The following is a sample Stream Out template file:
filename : streamOut.il

streamOutKeys = list(nil

'runDir "." ;- Run Directory
'libName "LIB" ;- Library Name
'primaryCell "" ;- Top Cell Name
&viewName "layout" ;- View Name
'outFile "test.gds" ;- Output File
'scale 0.001000 ;- Scale UU/DBU
'units "micron" ;"micron"/"milimeter"/"mil" - Units
'compression "none" ;- Compression
'hierDepth 32 ;(0-32) - Hierarchy Depth Limit
'convertToGeo "nil" ;t/n1 - Convert PCells to Geometry
'maxVertices 200 ;- Maximum Vertices in Path/Polygon
'refLib nil ;- Retain Reference Library (No Merge)
'libVersion "5.0" ;(3-6) - Library Version
'checkPolygon nil ;t/n1 - Report Bad Polygons
'snapToGrid nil ;t/n1 - Snap XY to Grid Resolution
'simMosaicToArray t ;t/n1 - Convert Simple Mosaic to Array
'caseSensitivity "upper" ;"preserve"/"upper"/"lower" - Case Sensitivity
'lineToZeroPath "path" ;"path"/"ignore" - Convert Lines to
'convertDot "node" ;"polygon"/"node"/"ignore"-Convert Dots to
'rectToBox nil ;t/n1 - Output Rectangle as BOX
'convertPathToPoly nil ;t/n1 - Convert Paths to Polygons
'keepPcell nil ;t/n1 - Keep PCells
'replaceBusBitChar nil ;t/n1 - Replace <> with []
'useParentXYforText nil ;t/n1 - To Use Parent XY for Text
'reportPrecision nil ;t/n1 - Precision Report
'runQuiet nil ;t/n1 - Filter Out Warning/Information Messages
'comprehensiveLog nil ;t/n1 - Comprehensive Log
'ignorePcellEvalFail nil ;t/n1 - Ignore Pcell evaluation failure
'errFile "PIPO.LOG" ;- Error Message File
'NOUnmappingLayerWarning nil ;t/n1 - Filter Out Unmapping Warning
'techFileChoice nil ;t/n1 - Techfile Choice
'pcellSuffix "DbId" ;"DbId"/"Params"/"DbIdPlusTime" - Pcell Suffix
'respectGDSIIlimits nil ;t/n1 - Respect GDSII limits
'dumpPcellInfo nil ;t/n1 - Dump Pcell Info
'genListHier nil ;t/n1 - Generate Hierarchy Listing
'cellMapTable "" ;- Cell Name Map Table
'layerTable "layerMap" ;- Layer Map Table
'textFontTable "" ;- Text Font Map Table
'convertPin "geometry" ;- Convert Pin to
'pinInfo 0 ;(0-127) - Keep pin information as attribute number
'pinTextMapTable "" ;- Pin Text Map Table
'propMapTable "" ;- User-Defined Property Mapping File
'propSeparator ";" ;- User-Defined Property Separator
'userSkillFile "" ;- User-Defined SKILL File
'rodDir "" ;- Rod Directory


For details on the above options, see Export – Stream Forms.

**Note:** Set the ‘gdsCompliantMosaic’ option to ‘t’ in the Stream Out template file to indicate to PIPO that the output stream file contains mosaics in the standard GDSII stream format as follows:

```
‘gdsCompliantMosaic nil ;t/nil
```

When this option is set to ‘nil’, you might find some placement differences in the way PIPO Stream Out translates or rotates mosaics to be compliant with the Stream In process. Default: ‘nil’. For details refer AREF Record Generation Behavior in PIPO-Deviation from the Stream Format on page 135.

## How Export – Stream Translates Data

There are important differences between GDSII Stream format and Design Framework II format. This section summarizes the major differences. To ensure the successful translation of your data, familiarize yourself with the format differences listed in this section. For a full description of the GDSII Stream format, refer to Appendix A, “Stream Format”.

The major differences between Stream format and Design Framework II format are the following:

- **Design Framework II format** supports only TEXT and SREF magnification. Stream permits TEXT, AREF, and SREF magnification.

- **Stream** does not support lines. *Export – Stream* translates lines to zero-width paths.

The following table shows the path-type translation.

<table>
<thead>
<tr>
<th>Design Framework II path type</th>
<th>GDSII Stream path type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dbcTruncateExtend(1)</code></td>
<td>0</td>
</tr>
<tr>
<td><code>dbcRoundRound(3)</code></td>
<td>1</td>
</tr>
<tr>
<td><code>dbcExtendExtend(2)</code></td>
<td>2</td>
</tr>
<tr>
<td><code>dbcVarExtendExtend(4)</code></td>
<td>4</td>
</tr>
</tbody>
</table>

- Because of the structural similarities between AREFs and simple mosaics, *Export – Stream* translates each simple mosaic to an AREF. However, for complex mosaics, each element is translated to an SREF.
Export – Stream Data Types

The following table shows how Export – Stream maps data types.

<table>
<thead>
<tr>
<th>Design Framework II data type</th>
<th>GDSII Stream data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>arc path</td>
<td>path</td>
</tr>
<tr>
<td>cell view</td>
<td>structure</td>
</tr>
<tr>
<td>circle</td>
<td>boundary</td>
</tr>
<tr>
<td>circle path</td>
<td>path</td>
</tr>
<tr>
<td>donut</td>
<td>boundary</td>
</tr>
<tr>
<td>dot</td>
<td>boundary or ignore</td>
</tr>
<tr>
<td>ellipse</td>
<td>boundary</td>
</tr>
<tr>
<td>instance</td>
<td>SREF</td>
</tr>
<tr>
<td>label</td>
<td>TEXT</td>
</tr>
<tr>
<td>library</td>
<td>library</td>
</tr>
<tr>
<td>Simple mosaic</td>
<td>AREF</td>
</tr>
<tr>
<td>Complex mosaic</td>
<td>SREFs</td>
</tr>
<tr>
<td>path</td>
<td>path</td>
</tr>
<tr>
<td>polygon</td>
<td>boundary</td>
</tr>
<tr>
<td>rectangle</td>
<td>boundary or BOX</td>
</tr>
<tr>
<td>symbolic devices</td>
<td>geometric data</td>
</tr>
</tbody>
</table>

Interpreting the Units Data

This section provides information you can use to interpret the units data descriptions.

The following table decodes units data on a hexadecimal level.

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Command follows</td>
</tr>
<tr>
<td>14...</td>
<td>20 (14 hexadecimal) bytes</td>
</tr>
</tbody>
</table>
To interpret the Stream eight-byte real data format, you need to look at its binary form:

```
SEEEEEEE .MMMMMMMM MMMMMMMM MMMMMMMM...MMMMMMMM
```

The bits have the following meanings:

- The sign (S) is 0 for positive numbers and 1 for negative numbers.
- To find the actual exponent, the seven-bit exponent (E...E) uses the excess-64 notation. Subtract 64 from the value. For example, if E...E is 0111110, which is 62 in decimals, then the exponent is 62-64=-2. This exponent is applied to 16^{-2}.
- The decimal point precedes the mantissa.
- The 56-bit mantissa expresses a binary fraction, that is, the first M represents 2^{-1} (or 1/2), the second represents 2^{-2} (1/4), and so on. Because significance drops off by nearly a factor of ten for every three bits, you can usually ignore the fourth through seventh bits of the mantissa.

To derive the units figure, multiply the mantissa by 16 raised to the exponent power.

The following table shows how to interpret the value of the first argument, using the standard-unit definition as an example.

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Binary</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
<td>Header for UNITS record, which takes two arguments</td>
<td></td>
</tr>
<tr>
<td>05</td>
<td>Each of the two arguments is in eight-byte real data format</td>
<td></td>
</tr>
<tr>
<td>3e 41...</td>
<td>First argument: the size of a database unit in user units</td>
<td></td>
</tr>
<tr>
<td>39 44...</td>
<td>Second argument: the size of a database unit in meters</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Binary</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3e 0011 1110</td>
<td>Sign and exponent byte</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Sign is zero: exponent is positive</td>
<td></td>
</tr>
<tr>
<td>011 1110</td>
<td>Exponent is 62 minus 64, or -2</td>
<td></td>
</tr>
<tr>
<td>41 0100 0001</td>
<td>First mantissa byte; 2**-2 + 2**-8...</td>
<td></td>
</tr>
<tr>
<td>89 1000 1001</td>
<td>...+ 2**-9 + 2**-16 = 0.256 (rounded)</td>
<td></td>
</tr>
<tr>
<td>37...ef</td>
<td>Least significant bytes of the mantissa</td>
<td></td>
</tr>
</tbody>
</table>
The value of this argument is 0.256 times 16 to the -2 power (1/256), or 0.001 database units per user unit.

The following table shows how to interpret the value of the second argument using the standard unit definition as an example.

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Binary</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>39</td>
<td>0011 1001</td>
<td>Sign and exponent byte</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Sign is 0: exponent is positive</td>
</tr>
<tr>
<td>011 1001</td>
<td></td>
<td>Exponent is 57 minus 64, or -7</td>
</tr>
<tr>
<td>44</td>
<td>0100 0100</td>
<td>First mantissa byte; $2^{-2} + 2^{-6}$...</td>
</tr>
<tr>
<td>b8</td>
<td>1011 1000</td>
<td>...$+ 2^{-9} + 2^{-11} + 2^{-12} + 2^{-13} = 0.256$ (rounded)</td>
</tr>
<tr>
<td>2f...51</td>
<td></td>
<td>Least significant bytes of the mantissa</td>
</tr>
</tbody>
</table>

The value of this argument is 0.268 times 16 to the -7 power or 0.000000001 database units per meter.

**Export – Stream Output Files**

Depending on what optional files you specify in the *Virtuoso® Stream Out* form or template file, *Export – Stream* produces one or more of the following files:

- A translated Stream file, with input objects mapped to output objects.
- A text version of the Design Framework II database file. When *Export – Stream* generates this file, it does not translate the Design Framework II database to Stream format.
- An error file (*PIPO.LOG* by default), which reports informational, warning, and error messages that might have occurred during translation.

**Error File**

The following is a sample Stream Out error file or log file. The *Export – Stream* parameter values used to produce this example are included at the end of the file.
1. translating cellview (a2a_gteq_64 layout)

WARNING (347): Failed to get Stream layer rules for layer-purpose pair ‘220:236’. Layer will be dropped.

2. translating cellview (a2a_lukahd4 layout2)
3. translating cellview (a2a_lukahd4_3 layout)

WARNING (415): Duplicate cell name found in design. Renaming ‘a2a_lukahd4’ to ‘a2a_lukahd4_3’.

4. translating cellview (pcon layout)
5. translating cellview (m12py layout)
6. translating cellview (pcon1 layout)
7. translating cellview (m12py2 layout)

Top Cells in File

a2a_gteq_64  layout

List Hierarchy (#cellInst,#arrayInst)

a2a_gteq_64  layout
  .a2a_lukahd4_3 layout(16)
    ..pcon1  layout(11)
    ..m12py  layout(6)
    ..m12py2 layout(1)
    ..pcon  layout(1)
  .a2a_lukahd4 layout2(4)
    ..pcon1  layout(11)
    ..m12py  layout(6)
    ..pcon  layout(2)
Individual Cell Statistics

<table>
<thead>
<tr>
<th>CellView</th>
<th>Rectangles</th>
<th>Polygons</th>
<th>Paths</th>
<th>Ellipses</th>
<th>Labels</th>
<th>Instances</th>
<th>Arrays</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>------------------------</strong></td>
<td><strong>-----------------------------------------------</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a2a_gteq_64/layout</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a2a_lukahd4/layout2</td>
<td>693</td>
<td>167</td>
<td>0</td>
<td>12</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a2a_lukahd4_3/layout</td>
<td>679</td>
<td>159</td>
<td>0</td>
<td>12</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pcon/layout</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m12py/layout</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>pcon1/layout</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m12py2/layout</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Statistics Of Layers

<table>
<thead>
<tr>
<th>Layer #</th>
<th>Rectangles</th>
<th>Polygons</th>
<th>Paths</th>
<th>Ellipses</th>
<th>Labels</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>36</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>37</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>41</td>
<td>44</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>42</td>
<td>152</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>606</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>51</td>
<td>191</td>
<td>0</td>
<td>182</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>53</td>
<td>98</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>55</td>
<td>89</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td>157</td>
<td>0</td>
<td>128</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>61</td>
<td>45</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Summary Of Objects

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1396</td>
<td>Rectangles</td>
<td>0</td>
<td>Polygons</td>
<td>326</td>
<td>Paths</td>
<td>0</td>
<td>Ellipses</td>
</tr>
<tr>
<td>24</td>
<td>Labels</td>
<td>58</td>
<td>Instances</td>
<td>0</td>
<td>Arrays</td>
<td>7</td>
<td>Cells</td>
</tr>
<tr>
<td>0</td>
<td>Nodes</td>
<td>0</td>
<td></td>
<td>00:00:01</td>
<td>Elapsed time</td>
<td>00:00:02</td>
<td>CPU time</td>
</tr>
</tbody>
</table>

July 2007 132 Product Version 5.1.41
29421 Kbytes of memory

*** There were 0 error and 2 warning messages ***

The setup file’s contents:
streamOutKeys = list(nil
  'runDir                   "."  
  'libName                  "terrym"  
  'primaryCell              "A2A_GTEQ_64"  
  'viewName                 "layout"  
  'outFile                  "Stream.1.out"  
  'scale                    0.001000  
  'units                    "micron"  
  'hierDepth                32  
  'convertToGeo             t  
  'maxVertices              200  
  'refLib                   nil  
  'libVersion               "5.0"  
  'checkPolygon             nil  
  'snapToGrid               nil  
  'simMosaicToArray         t  
  'caseSensitivity          "lower"  
  'lineToZeroPath           "path"  
  'rectToBox                 nil  
  'convertDot               "polygon"  
  'useParentXYforText       nil  
  'reportPrecision          nil  
  'runQuiet                 nil  
  'errFile                  "PIPO.LOG.1.out"  
  'NOUnmappingLayerWarning  nil  
  'cellMapTable             ""  
  'layerTable               ""  
  'textFontTable            ""  
  'convertPin               "geometry"  
  'pinInfo                  0  
  'pinTextMapTable          ""  
  'propMapTable             ""  
  'propSeparator             ""  
  'userSkillFile            ""  
  'rodDir                   ""  
)


ASCII Dump File

If you select **ASCII Dump File** in the **Virtuoso® Stream Out** form, **Export – Stream** produces a text file of the Stream file and the input file is not translated. The following is a sample ASCII dump file:

```
Begin Library
Library Name: main_lib, DB unit per user unit: 1000,
User Unit: Micron
Begin Cell Definition
Cell Name : CHIP_TOP, View Name : layout
Rectangle - Layer : 1 Data Type : 0 BBOX : (1000,1000) (3000,3000)
Polygon - Layer : 2 Data Type : 0 No of points : 9
(-5000,500) (-9500,500) (-9500,2500) (-8000,2500)
(-8000,4000) (-6500,4000) (-6500,3000) (-5000,3000)
(-5000,500)
Path-Layer: 3 Data Type: 0 No of points: 4 Width: 2000
Path Type: Truncate
(-2000,-500) (-2000,7000) (5000,7000) (5000,2000)
Cell Instance - Master Name : BLOCK1 Inst Name : I0
Origin: (9000,5500) Angle: 0.000 Mirror: 0 Magnification: 1.000
Array - Master Name : BLOCK2 Array Name : M0
    Origin: (-15000,-18500) Mirror: 0 Angle: 0.000
    Magnification: 1.000
    Rows : 2 Columns : 3 Row Spacing : 8500 Column Spacing : 15500
End Cell Definition
Begin Cell Definition
Cell Name : BLOCK1, View Name : layout
Rectangle - Layer : 1 Data Type : 0 BBOX :
(10500,1500) (12500,3500)
Polygon - Layer : 2 Data Type : 0 No of points : 9
(4500,1000) (0,1000) (0,3000) (1500,3000)
(1500,4500) (3000,4500) (3000,3500) (4500,3500)
(4500,1000)
Path-Layer: 3 Data Type: 0 No of points: 4 Width: 2000
Path Type: Truncate
(7500,0) (7500,7500) (14500,7500) (14500,2500)
End Cell Definition
Begin Cell Definition
Cell Name : BLOCK2, View Name : layout
Rectangle - Layer : 1 Data Type : 0 BBOX :
(10500,1500) (12500,3500)
Polygon - Layer : 2 Data Type : 0 No of points : 9
(4500,1000) (0,1000) (0,3000) (1500,3000)
(1500,4500) (3000,4500) (3000,3500) (4500,3500)
(4500,1000)
Path-Layer: 3 Data Type: 0 No of points: 4 Width: 2000
Path Type: Truncate
(7500,0) (7500,7500) (14500,7500) (14500,2500)
End Cell Definition
End Library
```
AREF Record Generation Behavior in PIPO-Deviation from the Stream Format

For rotated arrayed instances, the AREF records generated by PIPO in the output Stream file do not conform to the Stream format. This has been happening since many years and the data generated by PIPO has become an industry standard. PIPO Stream In as well as the third-party Stream readers recognize it as desired and are able to read this data correctly.

In Stream format, an AREF is represented as described below.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNAME</td>
<td>Name of the structure</td>
</tr>
<tr>
<td>COL</td>
<td>Number of columns</td>
</tr>
<tr>
<td>ROW</td>
<td>Number of rows</td>
</tr>
<tr>
<td>ANGLE</td>
<td>Angle of rotation</td>
</tr>
<tr>
<td>XY0</td>
<td>Origin point</td>
</tr>
<tr>
<td>XY1</td>
<td>Displacement from the origin by the inter-column spacing times the number of columns.</td>
</tr>
<tr>
<td>XY2</td>
<td>Displacement from the origin by the inter-row spacing times the number of rows.</td>
</tr>
</tbody>
</table>
Consider the following example of an arrayed instance.

The arrayed instance is placed at zero angle as shown in Figure 1. It is represented by the following information in the Stream format:

\[
\begin{align*}
\text{COL} &= 4 \\
\text{ROW} &= 3 \\
\text{ANGLE} &= 0 \\
\text{XY0} &= P0 (0,0) \\
\text{XY1} &= P1 (15000,0) \\
\text{XY2} &= P2 (0,8000)
\end{align*}
\]

PIPO writes this information correctly in the Stream file.

The arrayed instance is placed at zero angle as shown in Figure 1. It is represented by the following information in the Stream format:

\[
\begin{align*}
\text{COL} &= 4 \\
\text{ROW} &= 3 \\
\text{ANGLE} &= 0 \\
\text{XY0} &= P0 (0,0) \\
\text{XY1} &= P1 (15000,0) \\
\text{XY2} &= P2 (0,8000)
\end{align*}
\]

PIPO writes this information correctly in the Stream file.
Figure 2 shows the same arrayed instance rotated at an angle of 90 degrees. Notice that the points P0, P1, and P2 have moved to P0’, P1’, and P2’. In the Stream format, the coordinates should be represented by the following information:

- COL= 4
- ROW= 3
- ANGLE=90
- XY0 = P0 (0,0)
- XY1 = P1(15000,0)
- XY2 = P2(0,8000)

But PIPO writes the displaced points after rotation instead of writing the original points. Following information is written by PIPO:

- COL= 4
- ROW= 3
- ANGLE=90
- XY0 = P0’ (0,0)
- XY1 = P1’ (0, 15000)
- XY2 = P2’ (-8000, 0)

### Library Cell(s) Limits

In the previous releases, if a library consisted of 32765 (LINK_MAX) cells, no more cells could be added to it and the translation failed. In the present release, this problem has been fixed. Now, the target library is populated until LINK_MAX is reached. Subsequently, new libraries with the name targetlibname_PIPOLIB_number are created and PIPO uses them as a new libraries. The newly created PIPO generated libraries have technology data attached.
to the original target library. The generated log file contains TARGET LIBRARY PATH: as header for the newly created PIPO libraries.

```plaintext
************************************************************************
* CADENCE Design Systems, Inc.                                     *
* EXEC TIME : 28-Oct-2003 18:45:17                                  *
* @(#)CDS: pipo.exe version EXP 10/28/2003 16:38 (srishti) $          *
************************************************************************
Stream file : /tmp/LINK_MAX/BaseLib.gds

Reading Stream File ...
VERSION : 5
MODIFICATION : Tue Oct 28 09:27:34 2003
ACCESS : Tue Oct 28 09:31:05 2003
GDS LIBRARY : BASELIB.DB
U-UNIT/DBU : 0.001000000000
METRIC/DBU : 0.000000001000

TARGET LIBRARY PATH: /tmp/LINK_MAX/TARGETLIB
1. scanning cellview (Two layout)
2. scanning cellview (One layout)
3. scanning cellview (Base layout)

PASS 2...
1. translating cellview (One layout)
TARGET LIBRARY PATH: /tmp/LINK_MAX/TARGETLIB_PIPOLIB_1
2. translating cellview (Base layout)
......
```
Performance Guidelines for PIPO Stream In and PIPO Stream Out

PIPO Stream In and PIPO Stream Out provide many options to make its use model extremely flexible in accordance with the increasing requirements of the customers.

The following section provides information on the known speed impact of the PIPO Stream In and Stream Out options, and some guidelines on extracting maximum performance by avoiding those options that reduce the speed of translation in specific customer designs.

Guidelines for Stream In

1. In order to optimize the time consumed in searching for reference libraries during Stream In, use the methods described below:

   a. Reduce the cds.lib file size

   The cds.lib file should consist of a minimal list of libraries for Stream In to perform faster. The minimum number of libraries listed in cds.lib should be the target and reference libraries. The cells of these libraries should be instantiated in the stream file. If there are a large number of libraries mentioned in the cds.lib file, Stream In spends extra time in searching for different cellviews in those libraries and thus takes more time. Another way to resolve this issue without modifying the cds.lib file is by using the Reference Library Order option.

   b. Use the Reference Library Order option

   Suppose you are having a number of reference libraries in cds.lib and a design or a sub design/module is using a small fraction of the libraries that are being used. In another case, the libraries defined in cds.lib may not get ordered in an optimal fashion to minimize the search for a referenced cell during Stream In. In such a case, use the Reference Library Order option.

   The syntax of the Reference Library Order option in the template file is as follows:

   `refLibOrder "lib1 lib2 lib3 lib4"

   where lib1, lib2, lib3, and lib4 are the names of reference libraries separated by blank spaces.

   Example:

   Consider a situation where you are using instances from five different libraries: A, B, C, D, and E. You know that the usage density (the maximum probability of finding the referenced cell from a library) is from C and E. In such a case, in order to speed up the Stream In process, use the `refLibOrder option as shown below:
2. Avoid using the Keep Pcells and the Rod Directory options. These options might load some auxiliary SKILL files that are dumped during Stream Out, which take some extra time. Moreover, do not use these options if you do not intend to preserve the parameterized cells and ROD objects in your design during the round-trip of Stream Out and Stream In.

3. If the value of the Convert Array to Simple Mosaic option is chosen as off (a non-default value), it will explode each AREF record in a Stream file. This will be done by translating an AREF record to the corresponding individual instances in the dfl database. In some specific cases, this can reduce the translation speed and increase the output database size.

4. If a value of the Maximum Vertices in Path/Polygon option is chosen to be very low (such as 3 or 4), Stream In divides the stream polygons into many polygons in the dfl database. The division of stream polygons into many polygons can considerably reduce performance in certain designs.

5. Leave the Top Cell Name option empty if the stream file consists of only a top cell and its hierarchy. When the Top Cell Name option is left empty in the GUI, the corresponding value in the template file will be as shown below:

   ‘primaryCell " "

   This saves time marginally. If some value is specified for the primary cell, PIPO has to do some extra processing to identify the design hierarchy in the stream file, which is not actually desired for library translation.

6. Use the default value 32 for the Hierarchy Depth Limit option if you do not want to restrict the level of translation of the design. The value 32 is also the maximum limit of the level of translation. This value is set to 32 as shown below:

   ‘hierDepth 32

   The Hierarchy Depth Limit option is advantageous if you want to translate only up to the first few levels (up to 1 or 2) for specific purposes because the data translated in this case would be less. Incase of library translation, this option should be set to its default value. The value 20 for hierdepth was used in the 4.3.4 release before it was updated to 32 from the 4.4.1 release onwards. Therefore from 4.4.1 release, the use of 20 as the value results in extra processing (as the design may be within this limit) which is bound to marginally reduce the speed of translation during Stream In.

7. Avoid using the User-Defined SKILL File option because it slows down the translation speed during Stream In. The translation speed reduces because the User-Defined SKILL File option has to do extra processing for executing the given SKILL code each time to map the respective cell name, layer name, or text font for which it is being used. Therefore, use other options, such as the Layer Map Table, Cell Map Table, and Text Font Map Table options, wherever possible.
8. Use the Cell Map Table option only wherever it is specifically required as it can decrease the translation speed. This is because the extra processing requirements for each cell name can have considerable negative performance impact on Stream In. The negative performance impact is large in case of designs having a large number of cells compared to its actual size. For example, a 40 MB file having 5000 cells defined inside it will have a considerable negative performance impact on Stream In. This option can be used to handle special characters in the cell names that are not considered legal in some other tool or database in the design flow.

A case of unnecessary usage of this option could be to use a cell map file to create different names for distinguishing them.

For example, avoid the type of cell name map table shown below:

```
abc layout stream_abc
efg layout stream_efg
..........
```

9. Use the Skip PCDB Generation and Skip DB Locking options to enhance the speed of translation during Stream In. These options are accessible only through the template file of PIPO (that is its Command Line Interface) and have not been provided in the GUI. These options are available from the 5.0 release onwards, and are explained as follows:

- **Skip PCDB Generation**
  
  The Skip PCDB Generation option can be used to skip the .pc.db file generation if you wish to enhance the speed of translation further. If this option is not used, Stream In creates a .pc.db file in each cellview of the CDBA database by default. Therefore, this option also enables you to save some disk space.

  The Skip PCDB Generation option is expected to provide more speed especially for the designs containing a large number of cellviews in their hierarchy or library.

  **Caution**

  The absence of .pc.db files in the CDBA database might result in some problems while using the Hierarchy Browser tool or the Layout Versus Schematic (LVS) tool at times.

  The Skip PCDB Generation option can be activated by adding the following key in the input template file supplied during Stream In:

  ```
  'skipPcdbGen t
  ```

  The default value of this key is nil.

  Whenever you select the Skip PCDB Generation option while performing Stream In, you will get the following warning in the PIPO.LOG file as well as on the command prompt (stderr):
*Warning* The ‘skipPcdbGen’ option is chosen. The generation of ‘pc.db’ files in the output cellviews in the Cadence database will be skipped.

The Skip PCDB Generation option is enabled to work only if you are streaming to a new target library. If a target library already exists, this option will be automatically turned off with the following warning:

*Warning* The ‘skipPcdbGen’ option has been turned off because the target library ‘<libName>’ already exists.

where, <libName> refers to the target library that is of data type string.

❑ Skip DB Locking

The Skip DB Locking option can be used to skip the database locking completely and directly write data in the layout.cdb file to the CDBA database. The skipping of database locking and writing the layout.cdb data to the CDBA database reduces the run time of Stream In especially if the designs contain a large number of cellviews in their hierarchy or library.

Caution

If you are using the Skip DB Locking option, you will have to make sure that the target library is not accessed by another user simultaneously during Stream In. This option brings in chances of data corruption if a cellview is accessed by Stream In as well as by another user or any other process at the same time.

This option can be activated by adding the following key in the input template file supplied during Stream In:

'skipDbLocking t

The default value of this key is nil.

Whenever you select the Skip DB Locking option while performing Stream In, you will get the following warning in the PIPO.LOG file as well as on the command prompt (stderr):

*Warning* The ‘skipDbLocking’ option is chosen. The output cellviews in Cadence database will no longer be separately locked while being edited during Stream In.

The Skip DB Locking option is enabled to work only if you are streaming to a new target library. If the target library already exists, this option will be automatically turned off with the following warning:

*Warning* The ‘skipDbLocking’ option has been turned off because the target library ‘<libName>’ already exists.

where, <libName> refers to the target library that is of data type string.
Guidelines for Stream Out

1. Avoid the *Convert Pcells to Geometry* option. This option increases the Stream Out translation time because it has to flatten all pcell instances in the design.

2. Select the default value 32 for the *Hierarchy Depth Limit* option for library translation (not specifying the top cell name) if you do not want to restrict the level of translation of design hierarchy. If you select a non-default value for the *Hierarchy Depth Limit* option, it can considerably reduce the speed of translation. This is because Stream Out initially analyzes the entire design hierarchy for different cells in the dfll library.

   The *Hierarchy Depth Limit* option is useful if you want to translate only the first few levels (say 1 or 2), and the actual hierarchy is much deeper. In addition, the data translated would be much lesser when the *Hierarchy Depth Limit* option is used.

3. Avoid using the *Keep Pcells* and the *Rod Directory* options. These options result in dumping some auxiliary files during Stream Out. Moreover, do not use these options if you do not intend to preserve the parameterized cells and ROD objects in your design during the round-trip of Stream Out and Stream In.

4. If the value of the *Convert Array to Simple Mosaic* option is chosen as off (a non-default value), it will explode each simple mosaic in a Stream file. This will be done by translating each element of the mosaic to a corresponding SREF record in the Stream file. This can significantly reduce the translation speed and increase the size of an output Stream file.

5. Activate the *Dump Pcell Info* option only when a separate file containing information about the pcell variants is needed. In specific designs having pcells, if you select on, a non-default value for the *Dump Pcell Info* option (along with the pcell suffix chosen either as *Database ID* or *DbIDPlusTime*), an auxiliary data file *pipo_pcell_info* is generated in the run directory. This file provides extensive details about the pcell variants and their corresponding parameters. Because this file generation needs extra processing, it can have a considerable negative performance impact on Stream Out for designs having a large number of pcells.

6. For specific purposes, you can use a limited set of layers to be translated. This can be done by configuring the layer mapping table accordingly. Using the limited set of layers, you can speed up translation because the amount of data to be translated is reduced.

7. The selection of *Individual CellView* in the *Techfile Choice* option can also have a marginal decrease in speed.

8. Avoid using the *User-Defined SKILL File* option because it slows down the translation speed during Stream Out. The translation speed reduces because the *User-Defined SKILL File* option has to do extra processing for executing the given SKILL code each time to map the respective cell name, layer name, or text font for which it is being used.
Therefore, use other options, such as the *Layer Map Table*, *Cell Map Table*, and *Text Font Map Table* options, wherever possible.

9. Use the *Cell Map Table* option only wherever it is specifically required as it can decrease the translation speed. This is because the extra processing requirements for each cell name can have considerable negative performance impact on Stream Out. The negative performance impact is large in case of designs having a large number of cells compared to its actual size. For example, a 40 MB file having 5000 cells defined inside it will have a considerable negative performance impact on Stream Out. This option can be used to handle special characters in the cell names that are not considered legal in some other tool or database in the design flow.

A case of unnecessary usage of this option could be to use a cell map file to create different names for distinguishing them.

For example, avoid the type of cell name map table given below:

```
abc    layout    stream_abc
efg    layout    stream_efg
........
```

10. Avoid using the *Convert Paths to Polygons* option. This option converts each dfll path to a polygon (boundary record in Stream format) during Stream Out. If there are large number of paths in a design, this option can reduce the speed of translation due to the additional overhead of converting each path to a polygon internally.
Running 64 bit PIPO - Stream In/Stream Out

You may get a large Stream file of size greater than 2GB during Stream In or an output file of size greater than 2 GB during Stream Out. To work with such large files, you need to use 64-bit PIPO Stream In/Stream Out as they have high capacity.

By default, PIPO runs in the 32-bit mode.

While running, 32-bit PIPO detects if the Stream file size exceeds 2 GB and in such a case, if 64-bit PIPO is installed on the machine and the operating system on that machine is 64-bit, then the 64-bit PIPO automatically runs and generates Stream data. However, while running, if 32-bit PIPO detects that the stream file size exceeds 2 GB but either 64-bit PIPO is not installed on the machine or the operating system is not 64-bit, then 32-bit PIPO fails. In this case, you need to run 64-bit PIPO.

To use 64-bit Stream In/ Stream Out, you need to ensure the following setup

- The machine you are using must be a 64-bit machine with a 64-bit OS.
- 64-bit PIPO must be installed on the machine.
- The environment variable CDS_AUTO_64BIT must be set so as to include PIPO. For this, you can do one of the following:
  ```
  setenv CDS_AUTO_64BIT="ALL"
  setenv CDS_AUTO_64BIT="pio"
  setenv CDS_AUTO_64BIT="$CDS_AUTO_64BIT pipo"
  ```
  When this variable is set, if the operating system is 64-bit and if 64-bit PIPO is installed, then PIPO Stream In or Stream Out will automatically run 64-bit PIPO.

**Note:** It is recommended that you do not do the settings listed above unless you are sure of the large size of the Stream file. You should run 32-bit PIPO unless you are sure that the input Stream file size is larger than 2 GB or that the design is so large that it will generate a Stream file of size more than 2 GB. This is because there is a performance penalty when you run 64-bit PIPO.
Streaming Out Large Designs Using 32-Bit PIPO

PIPO runs in the 32-bit mode by default. However, 64-bit PIPO can be run for handling large Stream files. Refer to the section Running 64 bit PIPO - Stream In/Stream Out.

If you do not have 64bit PIPO installed or do not have a 64-bit machine with a 64-bit OS, then special mechanisms are required to stream out large designs. These mechanisms are as follows:

- Hierarchy Depth option

Use the Hierarchy Depth option to stream out the top cell to a stream file and then translate all the instance master cells in their separate stream files.

Consider the following example:

```
   A
  / \  \\
 B   C    D
 / \   /  \ \
E   F G    H
    \    \  \\
     G   H   I
```

Stream Out with options: Top Cell = A, hierdepth = 1 to get one stream file.
Stream Out with options: Top Cell = B to get another stream file
Stream Out with options: Top Cell = C to get another stream file
Stream Out with options: Top Cell = D to get another stream file

Using this method, for the above example, you will get four Stream files containing the entire design data.

- Layer Mapping

You can translate subsets of layers to get multiple Stream files. For this, you can use layer map files to map subsets of layers and then PIPO translates only those layers that are mapped.

For example, your design contains four layer-purposes (metal, drawing), (poly, drawing), (metal1, drawing), (diff, drawing), then you can create two layer map files. For more details, you can also refer to the section Layer name mapping file.

layerMapFile_1.txt
metal  drawing  0  0
metal1 drawing 11 0
layerMapFile_2.txt
poly drawing 1 1
diff drawing 22 1

Stream Out with option- layer map file layerMapFile_1.txt to get one stream file.
Stream Out with option- layer map file layerMapFile_2.txt to get another stream file.

Using this method, for the above example, you will get two Stream files containing the entire design data.
Keep PCell Permutations in Stream Out/Stream In

There can be four permutations for the Keep PCells setting in Stream Out and Stream In. These four permutations are as follows:

**OUT: Keep PCells t, IN: Keep PCells t**

In this case, the stream file will contain structs with names like `cellName$$dbId1`, `cellName$$dbId2`, `cellName$$dbId3` (one for each variant). In addition, we have pcell SKILL files dumped in the `KPDIR` directory located in the `run` directory. When read in, pcell structs are replaced by loading in the pcell SKILL files.

**OUT: Keep PCells t, IN: Keep PCells nil**

PIPO operation will be exactly the same as it is now. The only external difference is that the auxiliary external files will also be created on Stream Out. But we lose pcells in the database after Stream In, that is, they get replaced by their variants.

**OUT: Keep PCells nil, IN: Keep PCells t**

PIPO operation will be exactly the same as it is now (you lose pcells). Stream In is instructed to preserve PCells but there were no auxiliary output files created during Stream Out so Pipo has no information to preserve or recreate them when reading the stream data. User will get a warning in this case, that even though requested, pcells could not be preserved because of inadequate information.

**OUT: keepPcell nil, IN: keepPcell nil**

The Keep PCell option is not used here.
Translating CIF Files

This chapter describes the following:

- Overview on page 150
- Using Import – CIF on page 150
- How Import – CIF Translates Data on page 160
- Import – CIF Output Files on page 161
- Exporting with CIF on page 164
- How Export – CIF Translates Data on page 175
- Export – CIF Output Files on page 176
Overview


This chapter describes the following:

- How to use Import – CIF and Export – CIF
- How Import – CIF and Export – CIF translate data
- The Import – CIF and Export – CIF output files

Using Import – CIF

Import – CIF translates CIF mask data from CIF format into Design Framework II database format. In addition, Import – CIF checks geometric data, remaps layers, filters CIF files, and creates a text version of CIF data.

The following figure shows the files Import – CIF uses and generates. The required files are those that you must specify in the Virtuoso® CIF In form.

Note: You must specify a template file when you use the pipo command to execute Import – CIF. (The pipo command is the noninteractive, batch mode version of Import – CIF.)
Import – CIF

File ➔ Import ➔ CIF...

Translates files from CIF format to Design Framework II database format.

CIF In Form

User-Defined Data and Options

User-Defined Data displays the CIF In User-Defined Data form. This form lets you specify optional mapping tables and user-defined data, such as the User-Defined Property Mapping File, User-Defined Property Separator, and User-Defined Skill File.
**Options** displays the *CIF In Options* form. This form contains more parameters that help you specify how *Import – CIF* translates your CIF file and generates the error file.

**Template File** is a file you can use to load file names and option settings into the *Virtuoso® CIF In* form. Click *Load* to update the form with the values in the template file and click *Save* to save the current option settings to the template file. When you use *Import – CIF*, a template file is optional. However, if you use the *pipo* command, you must create and specify a template file.

**Run Directory** is the default directory for all the files you specify in the *Virtuoso® CIF In* form. If you want to read from or write to a file that is not in the run directory, type the full path name in the field for that file. Default: current directory (.)

**Input File** is the name of the CIF file you want to translate.

**Note:** It is not possible to import CIF ASCII files generated by using ASCII Dump.

**Top Cell Name** is the name of the cell at the top of the hierarchy that you want to translate and the name of the view for the cells you translate. The default view is *layout*. If you want to translate all cells in the CIF file to a layout view, leave the field empty. If you want the cells translated to a view other than *layout*, type the cell name and the view name in the field.

For example, assume the cell hierarchy is as follows:

```
   A
  / \  
 B   C
  /   /
 F   D  F
   /   
  E
```

The following table shows some examples of how you can translate cells.

<table>
<thead>
<tr>
<th>Cells to translate</th>
<th>DFII view to translate into</th>
<th>What to type in the field</th>
</tr>
</thead>
<tbody>
<tr>
<td>B F</td>
<td>layout</td>
<td>B</td>
</tr>
<tr>
<td>C D F E</td>
<td>layout3</td>
<td>C layout3</td>
</tr>
<tr>
<td>All cells</td>
<td>layout</td>
<td></td>
</tr>
<tr>
<td>All cells</td>
<td>layout3</td>
<td>* layout3</td>
</tr>
</tbody>
</table>

* This example assumes that you use the default value *layout* in the *View Name* field.
If you want **Import – CIF** to translate the cells in the hierarchy to different views, you need to create a cell name mapping table that specifies the cell names and views one by one. For more information about creating a cell mapping table, refer to the section “Cell Name Mapping File” on page 27.

**Output** specifies what type of file **Import – CIF** generates.

- **Opus DB** translates the CIF file into a Design Framework II database. You must specify a library name in the **Library Name** field for the Design Framework II database. You can also specify a technology file in the **ASCII Technology File Name** field for **Import – CIF** to use to compile the database. If you do not specify a technology file, **Import – CIF** uses the default technology file in the `samples/techfile` directory.

- **ASCII Dump** translates the CIF file to ASCII format. If you select this option, you must specify a filename in the **ASCII Dump File** field. If you set this option on, **Import – CIF** does not generate a Design Framework II database.

  **ASCII Dump File** is available when you set the **Output** option to **ASCII Dump**. This is the file in which **Import – CIF** writes a text interpretation of the Design Framework II database binary file.

- **TechFile** extracts the layers information from the input CIF file and creates an ASCII technology file. **Import – CIF** does not generate a Design Framework II database when this option is on. When you set this option on, you must type a filename in the **ASCII Technology File Name** field.

  **ASCII Technology File Name** is the name of the technology file you want to create when you set the **Output** option to **TechFile**. If you choose **Output** option **Opus DB**, **Import – CIF** uses the technology file you specify here to create the Design Framework II database.

**Library Name** is the Design Framework II database file in which **Import – CIF** writes the translated APPLE 860 file. When you set the **Output** option to **Opus DB**, **Import – CIF** uses this field.

**Scale UU/DBU** is the ratio of database units per user unit you want to use in the Design Framework II file. Default: 0.010

For information on scaling your design, you can refer to the section “Scaling Your Design” on page 24.

**Units** is the unit of measure you want to use in the Design Framework II file. CIF data does not specify user units in its data format.

- **micron** sets the unit to one micron (0.001 millimeter).
- **millimeter** sets the unit to one millimeter.
mil sets the unit to one mil (approximately 25.4 microns).

**Process Nice Value 0-20** lets users adjust the “nice” priority value. The nice priority value is a factor used by the UNIX operating system to schedule the CPU time allocated to a process. The larger the value, the longer it will take *Import – CIF* to translate your design. Valid Values: an integer between 0 and 20, with 0 being highest priority. Default: 0

**Error Message File** is the error file where *Import – CIF* writes all warning, information, and error messages that occur as it translates data. Default: **PIPO.LOG**

### CIF In User-Defined Data Form

![CIF In User-Defined Data Form](image)

**Cell Name Map Table** is an optional file you can use to map cell names from the CIF file to the Design Framework II database. You can also specify cell name mapping information in a SKILL file. You specify the SKILL file in the *CIF In User-Defined Data* form. If you do not provide cell mapping information, *Import – CIF* uses the *Case Sensitivity* option setting from the *CIF In Options* form.

For a flow diagram of how *Import – CIF* determines how to translate a cell name, refer to the diagram in the section “Using Map Files When Importing” on page 26. For detailed information about how to create a cell name map table, refer to the section “Cell Name Mapping File” on page 27.

**Layer Map Table** is an optional file you can use to map CIF layers to Design Framework II database layer-purpose pairs (LPP). This can be useful when you change layer numbers, merge different layers into one layer, and filter layers. You can also specify layer mapping information in a SKILL file. You specify the SKILL file in the *CIF In User-Defined Data* form. If you specify a layer map table, *Import – CIF* does not check a SKILL file for layer mapping.

For detailed information about how to create a layer map table, refer to the section “Layer Name Mapping File” on page 28.
Text Font Map Table is an optional file you can use to map CIF fonts and text sizes to Design Framework II fonts. You can also specify text font mapping information in a SKILL file. You specify the SKILL file in the CIF In User-Defined Data form. If you specify a text font map table, Import – CIF does not check a SKILL file for font mapping.

For more information about how to create a text font map table, refer to the section “Text Font Mapping File for Stream In” on page 32.

User-Defined Skill File is the file containing user-defined SKILL routines that the system uses to translate cell names, layer names, text fonts, and illegal shapes. For information about writing SKILL routines refer to the SKILL Language User Guide and SKILL Language Reference.

CIF In Options Form

<table>
<thead>
<tr>
<th>CIF In Options</th>
<th>OK</th>
<th>Cancel</th>
<th>Defaults</th>
<th>Apply</th>
<th>Help</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report Bad Polygons</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Snap XY to Grid Resolution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Skip Undefined Layer-Purpose Pair</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convert Zero-Width Wire to</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convert Round Rash to</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Case Sensitivity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convert Wire to Path Style</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Precision Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Save Cell Views At The End</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Do Not Overwrite Existing Cell</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Filter Out Warning/Information Messages</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Filter Out Unmapping Warning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hierarchy Depth Limit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maximum Vertices in Path/Polygon</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comprehensive Log</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ignore Pcell evaluation failure</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

32

1024
Design Data Translator's Reference
Translating CIF Files

**Report Bad Polygons** determines whether Import – CIF removes collinear and coincident points in polygons and paths and reports incomplete or illegal polygons in the error file. A polygon can have more than one error, although Import – CIF reports only the most severe problem in the error file. Import – CIF reports the following problems, listed here in order of severity with the most severe problem described first.

- **Open**: An open polygon is one whose first and last coordinates do not coincide.
- **Reentrant**: A reentrant polygon is one in which the sum of the internal angles is not 360 degrees or one in which one or more pairs of edges intersect rather than meet.
- **Acute angle**: An acute angle is an angle that is smaller than 90 degrees.
- **Non-45 degree angle**: A non-45 degree angle is an angle that is not a multiple of 45 degrees. Valid angles are 90, 135, 180, 225, 270, 315, 360.

**Snap XY to Grid Resolution** adjusts the coordinates of geometric objects to coincide with the grid resolution. When you use this option, you can use graphic commands (such as the Virtuoso Layout Editor commands *Move, Stretch, and Split*) to access end points, edges, and vertexes of geometric objects in your translated file. However, this option can shift data from the original grid, resulting in a loss of precision.

**Note**: You can set the grid resolution using `mfgGridResolution` in the Physical Rules class of the technology file you specify in the *ASCII Technology File* field. If you do not define `mfgGridResolution`, Import – CIF sets it to the inverse of the value you set for the *Scale UU/DB* field.

**Skip Undefined Layer-Purpose Pair** skips all the layer-purpose pairs that you do not define in the technology file for the current design library. You can use this option to drop certain layers of the input data in the translation.

Default: off

**Convert Zero-Width Wire to** determines whether Import – CIF translates zero-width wires to polylines in the Design Framework II database.

- **lines** converts zero-width paths to polylines.
- **ignore** does not translate zero-width paths.

**Convert Round Flash to** indicates that you want to translate round flashes to ellipses or polygons.

- **ellipse** converts round flashes to ellipses.
- **polygon** converts round flashes to polygons.
**Case Sensitivity** specifies whether *Import – CIF* changes the case of letters in cell names. *Import – CIF* uses the setting of this option when you do not supply cell name mapping information for the cell. You can map cell names in a cell name map table or in a SKILL file.

- **preserve** converts cell names without changing case. For example, the name *Obj1* remains *Obj1*.
- **upper** converts all cell names to uppercase. For example, the name *Obj1* becomes *OBJ1*.
- **lower** converts all cell names to lowercase. For example, the name *Obj1* becomes *obj1*.

**Convert Wire to Path Style** specifies how you want *Import – CIF* to translate path ends.

- **extend** translates wires to paths with ends extended by a distance equal to half the path width.

![Diagram](image)

- **round** translates wires to paths with ends extended by a half-circle of radius equal to half the path width.

![Diagram](image)

- **truncate** translates wires to paths with unextended ends.

**Precision Report** indicates whether *Import – CIF* checks precision and writes a warning message to the error file if precision is lost during translation. A loss of precision occurs when *Import – CIF* rounds a coordinate value so that it falls off grid. If you set this option on,
*Import – CIF* checks precision and writes warnings to the error file. When you set this option on, *Import – CIF* runs more slowly. If you set this option off, *Import – CIF* does not check precision. Default: off

**Save Cell Views At The End** saves the translated cellviews to disk at the end of the translation. If you set this option off, *Import – CIF* saves each cellview to disk as it is translated. If you set this option on, *Import – CIF* stores all cellviews in memory until it completes the translation, then *Import – CIF* stores them to disk. If the input file is smaller than 5 Mbytes, you can turn this option on to speed up the translation. Default: off

**Not Overwrite Existed Cell** prevents any cell that already exists in the library from being overwritten. Default: off

**Filter out warning/information messages** indicates whether *Import – CIF* writes warning and information messages to the error file during the translation process. If you set this option on, *Import – CIF* writes only error messages to the error file. If you set this option off, *Import – CIF* writes all messages to the error file. Default: off

**Filter Out Unmapping Warning** lets you choose whether *Import – CIF* writes warning messages about Cadence layers to the error file during the translation. Warnings about user layers will not be affected. Default: off

**Hierarchy Depth Limit** is the maximum number of levels of the input file hierarchy that you want the system to translate. Default: 20

**Maximum Vertices in Path/Polygon** lets you specify the maximum number of vertexes you want in polygons and paths in the Design Framework II database. *Import – CIF* divides polygons and paths that have more than this number of vertexes into smaller objects. Default: 1024. Valid Value: a positive integer from 1 to 1024

**Comprehensive Log** indicates whether *Import – CIF* writes warning and error messages generated by SKILL function calls and PIPO dependencies to the PIPO log file during translation. Default: nil

**Ignore Pcell Evaluation Failure** lets you choose whether *Import-CIF* should ignore pcell evaluation failures or stop translation on encountering a pcell evaluation failure. If you do not select this option, *Import-CIF* stops translation when it encounters a pcell evaluation failure and a fatal message is displayed. Default: nil

**Importing a CIF File from a Form**


The Virtuoso® CIF In form appears.
2. Type the CIF file to translate.
3. Type the output library name.
4. Specify any optional fields you want.
5. Click Apply or OK.

*Import – CIF* begins the conversion.

### Importing a CIF File from the Command Line

1. Create a template file using the *Virtuoso® CIF In* form.

   For information about how to create the template file, see the section “Preparing a Template File and .cdsinit File” on page 159.

2. Generate a technology file by entering the following command line:

   ```
   pipociftechgen templateFile
   ```

3. Type the following at the command line:

   ```
   pipocifin templateFile
   ```

### Preparing a Template File and .cdsinit File

A collection of option values that you can load into the *Virtuoso® CIF In* form can be defined in a template file or in the .cdsinit file. The required values in the CIF In template file and the .cdsinit file are exactly the same. Users can use either way to load the option values into the *Virtuoso® CIF In* form. A template file can be loaded into the *Virtuoso® CIF In* form in interactive mode or specified when you use the `pipocifin` command while the .cdsinit file is automatically loaded into *Virtuoso® CIF In* form when `layout` is activated. You can create a template file in one of two ways:

- Enter values in the *Virtuoso® CIF In*, *CIF In Options*, and *CIF In User-Defined Data* forms and click the Save command to save the option values to the file you specify as the template file.

- Create a text file. You can copy the sample template file `cifIn.il` in the `samples/transUI` directory and modify the file.

The required values in the *Import – CIF* template file are the name of the input file and the name of the Design Framework II library. The other values are optional. The following is a sample CIF In template file:

```lisp
    cifInKeys = list(nil
         'runDir ":"'
         'inFile '" ;CIF file will convert to Cadence
```
'libName "" ;the library
;
; generate technology file
; to generate new techfile - use keyword "strmtechgen"
; and specify techfile name
; to load techfile - just specify file name.
'techfileName "" ;output technology file
;
;input CIF specify
;
;scale 0.01
'units "micron" ;("micron","millimeter","mil")
;
; select CIF data
;
'primaryCell ""
'hierDepth 20
;
; mapping CIF data to Cadence
;
'caseSensitivity "lower" ;("lower","upper","preserve")
'cellMapTable ""
'layerTable ""
'textFontTable ""
'skipUndefined LPP nil
'noWriteExistCell ;(t/nil)
;
; check CIF input data
;
'checkPolygon nil ;(t/nil)
'snapToGrid nil ;(t/nil)
'errFile ""
'dataDump ""
'reportPrecision nil
'runQuiet nil
'NOUnmappingLayerWarning ;(t/nil)
;
; unmatch Cadence data type handle
;
'zeroWireToLine "line" ;("line","ignore")
'wireToPathStyle "extend" ;("extend","round","truncate")
'roundFlashToEllipse "ellipse" ;("ellipse","polygon")
;
; user SKILL file specified here
;
'userSkillFile ""
)

How Import – CIF Translates Data

There are important differences between CIF format and Design Framework II database format. To ensure successful translation of your CIF data, familiarize yourself with the information in this section.

The major differences between CIF format and Design Framework II format are the following:
Design Framework II format does not support absolute angles. Import – CIF translates only relative-angle-placed text, instances, and arrays; it discards the information dealing with absolute angles and issues a warning message.

Design Framework II format supports only 90-degree rotations. CIF permits rotation at any angle. The following table shows the angle translations.

<table>
<thead>
<tr>
<th>CIF Angle Format</th>
<th>Reflection</th>
<th>Design Framework II Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>0≤ and ≤45</td>
<td>yes</td>
<td>MY</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>R0</td>
</tr>
<tr>
<td>&gt;45 and ≤135</td>
<td>yes</td>
<td>MYR90</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>R90</td>
</tr>
<tr>
<td>&gt;135 and ≤225</td>
<td>yes</td>
<td>MX</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>R180</td>
</tr>
<tr>
<td>&gt;225 and ≤315</td>
<td>yes</td>
<td>MXR90</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>R270</td>
</tr>
<tr>
<td>&gt;315 and ≤360</td>
<td>yes</td>
<td>MY</td>
</tr>
<tr>
<td></td>
<td>no</td>
<td>R0</td>
</tr>
</tbody>
</table>

CIF format does not have LINE. Design Framework II format does not support zero-width PATH. Import – CIF translates zero-width paths to lines. The following table shows how Import – CIF maps data types.

<table>
<thead>
<tr>
<th>CIF Data Type</th>
<th>Design Framework II Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>polygon</td>
<td>polygon</td>
</tr>
<tr>
<td>round flash</td>
<td>polygon or ellipse</td>
</tr>
<tr>
<td>wire</td>
<td>path</td>
</tr>
<tr>
<td>call symbol</td>
<td>instance</td>
</tr>
<tr>
<td>box</td>
<td>rectangle</td>
</tr>
</tbody>
</table>

Import – CIF Output Files

Depending on what output files you specified in the run form or template file, Import – CIF produces one or more of the following files:

- A translated design data file, with input objects mapped to output objects.
A text version of the APPLE 860 database file. When *Import – CIF* generates this file, it does not translate the input file to Design Framework II format.

An ASCII technology file. When *Import – CIF* generates this file, it does not translate the APPLE 860 database file to Design Framework II format.

An error file (`PIPO.LOG` by default) that reports statistics and any warnings that might have occurred during translation.

### Error File

The following is an example of a log file that results from running *Import – CIF*. The template file used to generate this log file follows.

```
*******************************************************************
PIPO Summary Report File
*******************************************************************
READING CIF FILE...
CIF FILE: /usr1/demo/cif/shapes.cif
1. translating cellview (polygon layout)
2. translating cellview (storm layout)
3. translating cellview (flashes layout)
4. translating cellview (stormflash layout)
5. translating cellview (boxes layout)
6. translating cellview (wires layout)
7. translating cellview (labels layout)
8. translating cellview (shapes layout)

Saving all cells...
Top Cells in File
*******************************************************************
shapes layout
List Hierarchy(#cellInst,#arrayInst)
*******************************************************************
Individual Cell Statistics
shapes layout
  .labels layout(1)
  .wires layout(1)
  .boxes layout(2)
  .stormflash layout(1)
  ..flashes layout(1)
  ..storm layout(1)
  ...polygon layout(8)
```

<table>
<thead>
<tr>
<th>CellView</th>
<th>Rectangles</th>
<th>Polygons</th>
<th>Paths</th>
<th>Ellipses</th>
<th>Labels</th>
<th>Instances</th>
<th>Arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>polygon/layout</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>storm/layout</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>flashes/layout</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>stormflash/layout</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Design Data Translator's Reference
Translating CIF Files

boxes/layout   5  0  0  0  0  0  0
wires/layout   0  0  7  0  0  0  0
labels/layout  0  0  0  0  7  0  0
shapes/layout  0  0  1  0  0  5  0

Statistics of Layers

Layer-#  Rectangles  #Polygons  #Paths  #Ellipses  Labels
*******************************************************************
1        0           1           0       0           0
2        0           0           0       4           0
3        1           0           0       0           0
4        1           0           0       0           0
5        1           0           0       0           0
6        1           0           0       0           0
7        1           0           0       0           0
8        0           0           1       0           0
9        0           0           1       0           0
10       0           0           5       0           0
11       0           0           0       0           7
12       0           0           1       0           0

Summary of Objects

*******************************************************************
5          Rectangles
1          Polygons
8          Paths
4          Ellipses
7          Labels
15         Instances
0          Arrays
8          Cells
00:00:12 Elapsed Time
00:00:05 CPU time
1528 Kbytes of memory

*** There were no error or warning messages ***
NORMAL EXIT...

Template File

The template file contains

cifInKeys = list(nil
  'runDir "."
  'inFile "shapes.cif"
  'libName "shLib"
  ;techfileName "cifTF.out"
  'scale 1.00
  'units "micron"
  'primaryCell ""
  'hierDepth 20
  'caseSensitivity "lower"
  'cellMapTable ""
  'layerTable ""
ASCII Dump File

If you select ASCII Dump in the Virtuoso® CIF Input, Import – CIF produces an ASCII text image file of the input CIF file. If an ASCII dump file is generated, the CIF file is not translated.

The following is an example of an ASCII dump file:

```
Begin Library

Library Name: uclib, DB unit per user unit : 100, User Unit : micron
Begin Cell Definition
Cell Name : BOXES, View Name : layout
Rectangle - Layer : 7 Data Type : 0 BBOX : (-55,45) (-45,75)
Rectangle - Layer : 6 Data Type : 0 BBOX : (-65,35) (-35,85)
Rectangle - Layer : 5 Data Type : 0 BBOX : (-75,25) (-25,95)
Rectangle - Layer : 4 Data Type : 0 BBOX : (-85,15) (-15,105)
Rectangle - Layer : 3 Data Type : 0 BBOX : (-95,5) (-5,115)
End Cell Definition
End Library
```

Exporting with CIF

Export – CIF translates files from Design Framework II database format to CIF mask data format. In addition, Export – CIF checks geometric data, remaps layers, and creates a text version of Design Framework II data.
The following figure shows the files *Export – CIF* uses and generates. The required files are those that you must specify in the Virtuoso® CIF Out form.

**Note:** You must specify a template file when you use the `pipo` command to execute *Export – CIF*. (The `pipo` command is the noninteractive, batch mode version of *Export – CIF*.)
Export – CIF

File ➔ Export ➔ CIF...

Translates files from Design Framework II format to CIF format.

CIF Out Form

User-Defined Data and Options

User-Defined Data displays the CIF Out User-Defined Data form. This form lets you specify optional mapping tables and user-defined data, such as the User-Defined Property Mapping File, User-Defined Property Separator, and User-Defined Skill File.
Options displays the CIF Out Options form. This form contains more parameters that help you specify how Export – CIF translates your Design Framework II database file. The CIF Out Options form also contains parameters that let you filter the messages that Export – CIF writes to the error file.

Template File is the setup file that supplies filenames and option settings for the CIF Out form. Click Load to update the form with the values in the template file and click Save to save the current option settings to the template file. When you use Export – CIF, a template file is optional. However, if you use the pipo command, you must create and specify a template file.

Run Directory is the default directory for all the files you specify in the Virtuoso® CIF Out form. If you want to read from or write to a file that is not in the run directory, type the full path name in the field for that file. Default: current directory (.)

Library Name is the Design Framework II database library you want to translate to CIF.

Top Cell Name lists the name of the cell at the top of the hierarchy that you want to translate, the name of the view that you want to translate, and the version number of the view. If you omit the view name, Export – CIF uses the view name specified in the View Name field. Export – CIF uses the version number for the top cellview only. All subsequent cellviews in the hierarchy become the latest version in the library. If you do not specify a version number for the top cell, Export – CIF uses the latest version.

You can enter multiple top cell views. Use a semicolon (;) to separate each top cellview and use a blank space to separate the cell name and the view name.

The syntax is

```
cellA viewX ver1; cellB viewY ver2; ...
```

You can enter as many top cells as you want. The default value is an empty field, which translates the entire library.

For example, assume the cell hierarchy is as follows:

```
        A
       / \
      B   C
     / \ / \
    E   D G H
   / \ / \ / \
  F   H  I
```

July 2007

167

Product Version 5.1.41
The following table shows some examples of how you can translate cell hierarchies.

<table>
<thead>
<tr>
<th>Top cells to Translate</th>
<th>View to Translate</th>
<th>Version to Translate</th>
<th>What to type in the field</th>
</tr>
</thead>
<tbody>
<tr>
<td>B D</td>
<td>layout*</td>
<td>latest</td>
<td>B;D</td>
</tr>
<tr>
<td>B C</td>
<td>layout2, layout*</td>
<td>1.2, latest</td>
<td>B layout2 1.2;C</td>
</tr>
<tr>
<td>C</td>
<td>layout*</td>
<td>2.1</td>
<td>C 2.1</td>
</tr>
<tr>
<td>All cells</td>
<td>layout*</td>
<td>latest</td>
<td>* layout3 3.0</td>
</tr>
</tbody>
</table>

* This example assumes that you use the default value layout in the View Name field.

View Name is the default view name for the top cells specified in Top Cell Name. Export – CIF uses this view name only for the top cells for which you do not specify a view name. Default: layout

Output selects the output data format that Export – CIF creates.

- CIF DB translates the Design Framework II library data to CIF format.
- ASCII Dump translates the Design Framework II library to ASCII format. If you select this option, Export – CIF does not translate the Design Framework II database. When you set this option on, you must specify a filename in the ASCII Dump File field.

Output File is the name of the CIF file to which to translate the Design Framework II database file. You must type an output file if you selected CIF DB as your Output option.

ASCII Dump File is available when you set the Output option to ASCII Dump. This is the file in which Export – CIF writes a text interpretation of the Design Framework II database binary file.

Scale UU/DBU is the ratio of database units per user unit you want to use in the output file. Default: 0.01

For information on scaling your design, refer to the section "Scaling Your Design" on page 24.

Units is the basic unit for lengths that you want to use in the output file.

- micron sets the unit to one micron (0.001 millimeter).
- millimeter sets the unit to one millimeter.
Design Data Translator's Reference
Translating CIF Files

- **mil** sets the unit to one mil (approximately 25.4 microns).

For example, if the input file units are in microns, but you want to translate the units to mil in the CIF file, set this option to **mil**. The coordinate 254 in input data becomes 10 in output data (254 microns = 10 mils). If the Design Framework II file units are in mil, and you set this option to **micron**, the coordinate 20 becomes 508 in output data (20 mils = 508 microns).

**Process Nice Value 0-20** lets users adjust the “nice” priority value. The nice priority value is a factor used by the Unix operating system to schedule the CPU time allocated to a process. The larger the value, the longer it will take Export – CIF to translate your design. Valid Values: an integer between 0 and 20, with 0 being highest priority. Default: 0

**Error Message File** is the name of the file where Export – CIF writes information, warning, and error messages. Default: **PIPO.LOG**

**CIF Out User-Defined Data Form**

![CIF Out User-Defined Data Form](image)

**Convert Pin to** either translates pins into cell instances or labels or ignores them. If you select **text**, you must specify a pin text map table.

- **geometry** translates pins into cell instances.
- **text** translates pins into labels.
- **geometry & text** translates pins into shapes or cell instances and labels.
- **drop** does not translate pins.

**Pin Text Map Table** is the file you use to map the types of pins in your design to labels. You must specify this file when you set **Convert Pin to text**.
For more information about how to create a property mapping file, refer to the section “Pin Text Mapping File for Stream Out” on page 44.

**Note**: You must specify a file when you set *Convert Pin to text*. If all the pins in your design are geometry pins, you must specify a filename, even if the file is empty.

**Cell Name Map Table** is an optional file that you can use to map cell names from the Design Framework II database to the output files. You can also specify cell name mapping information in a SKILL file. You specify the SKILL file in the *CIF Out User-Defined Data* form. If you do not provide cell mapping information, *Export – CIF* uses the Case Sensitivity option setting from the *CIF Out Options* form.

For details about how *Export – CIF* translates cell names, refer to the diagram in the section “Using Map Files when Exporting” on page 36. For detailed information about how to create a cell name map table, refer to the section “Cell Name Mapping File for Stream Out” on page 37.

**Layer Map Table** is an optional file that tells *Export – CIF* how to map Design Framework II database layer-purpose pairs to CIF layers. This is useful for changing layers, merging different layers into one layer, and filtering layers. You can also specify layer mapping information in a SKILL file. You specify the SKILL file in the *CIF Out User-Defined Data* form. If you specify a layer map table, *Export – CIF* does not check a SKILL file for layer mapping.

For detailed information about how to create a layer map table, refer to the section “Layer Name Mapping File for Stream Out” on page 37.

**User-Defined Skill File** is the file that contains user-defined SKILL routines that the system uses to map cell names, parameterized cell names, layer names, and text fonts and to translate illegal shapes. For information about writing SKILL routines, see “Customizing Export Files” on page 279 or refer to the *SKILL Language User Guide* and *SKILL Language Reference* for instructions on using the SKILL language.
### CIF Out Options Form

<table>
<thead>
<tr>
<th>Option</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dot to Polygon</td>
<td>off</td>
</tr>
<tr>
<td>Report Bad Polygons</td>
<td></td>
</tr>
<tr>
<td>Snap XY to Grid Resolution</td>
<td></td>
</tr>
<tr>
<td>To Use Parent XY for Text</td>
<td></td>
</tr>
<tr>
<td>Convert PCells to Geometry</td>
<td></td>
</tr>
<tr>
<td>Convert Line to</td>
<td></td>
</tr>
<tr>
<td>Convert Magnification Cell to</td>
<td></td>
</tr>
<tr>
<td>Case Sensitivity</td>
<td></td>
</tr>
<tr>
<td>Precision Report</td>
<td></td>
</tr>
<tr>
<td>Filter Out Warning/Information Messages</td>
<td></td>
</tr>
<tr>
<td>Filter Out Unmapping Warning</td>
<td></td>
</tr>
<tr>
<td>Maximum Vertices in Path/Polygon</td>
<td>514</td>
</tr>
<tr>
<td>Comprehensive Log</td>
<td></td>
</tr>
<tr>
<td>Ignore Pcell evaluation failure</td>
<td></td>
</tr>
</tbody>
</table>

**Dot to Polygon** indicates whether you want to translate dot objects to polygons. When you set this option on, **Export – CIF** translates dots to polygons. When you set this option off, **Export – CIF** translates rectangles to polygons. Default: **off**

**Report Bad Polygons** determines whether **Export – CIF** removes collinear and coincident points in polygons and paths and reports incomplete or illegal polygons in the error file. A polygon can have more than one error, although **Export – CIF** reports only the most severe problem in the error file. **Export – CIF** reports the following problems, listed here in order of severity with the most severe problem described first.

- **Open**: An open polygon is one whose first and last coordinates do not coincide.
- **Reentrant**: A reentrant polygon is one in which the sum of the internal angles is not 360 degrees or one in which one or more pairs of edges intersect rather than meet.
- **Acute angle**: An acute angle is an angle that is smaller than 90 degrees.
- **Non-45 degree angle**: A non-45 degree angle is an angle that is not a multiple of 45 degrees. Valid angles are 90, 135, 180, 225, 270, 315, 360.
Snap XY to Grid Resolution adjusts the coordinates of geometric objects to coincide with the grid resolution. This lets graphic commands (such as the Virtuoso Layout Editor commands Move, Stretch, and Split) access end points, edges, and vertexes of geometric objects. However, this option can shift data from the original grid, resulting in a loss of precision.

Export – CIF uses the view property minGridResolution from the Design Framework II library technology file to scale the grid resolution. If you do not define this property, Export – CIF uses a value of 1 for the minGridResolution property when it translates your data. Default: off

To Use Parent XY for Text controls the position of text. When set on, uses as its origin the point at which the text attaches to the shape. Default: off

Convert PCells to Geometry flattens parameterized cell instances in the design to create polygon layout cellviews. Default: off

Convert Lines to indicates whether you want Export – CIF to translate polylines in the Design Framework II database to zero-width paths in the CIF file.

- wire translates polylines to zero-width paths.
- ignore does not translate polylines.

Convert Magnification Cell to indicates whether you want Export – CIF to translate magnified cells to polygons.

- polygon translates magnified cells to polygons.
- ignore does not translate magnified cells.

Case Sensitivity specifies whether Export – CIF changes the case of letters in cell names. Export – CIF uses the setting of this option when you do not supply cell name mapping information for the cell. You can map cell names in a cell name map table or in a SKILL file.

- preserve translates cell names without changing case. For example, the name Obj1 remains Obj1.
- upper translates all cell names to uppercase. For example, the name Obj1 becomes OBJ1.
- lower translates all cell names to lowercase. For example, the name Obj1 becomes obj1.

Precision Report indicates whether Export – CIF checks precision and writes a warning message to the error file if precision is lost during translation. A loss of precision occurs when Export – CIF rounds a coordinate value so that it falls off grid. If you set this option on,
Export – CIF checks precision and writes warnings to the error file. When you set this option on, Export – CIF runs more slowly. If you set this option off, Export – CIF does not check precision. Default: off

Filter out warning/information messages indicates whether Export – CIF writes warning and information messages to the error file during translation. If you set this option on, Export – CIF writes only error messages to the error file. If you set this option off, Export – CIF writes all messages to the error file. Default: off

Filter Out Unmapping Warning lets you choose whether Export – CIF writes warning messages about Cadence layers to the error file during the translation. Warnings about user layers are not affected. Default: off

Maximum Vertices in Path/Polygon indicates the maximum number of vertexes you want to allow in a polygon. The current limitation is 512. Export – CIF cuts any polygon or path that contains more than 512 vertexes to smaller ones. Export – CIF outputs a warning message to the error file for each cut polygon and path.

Note: Polygons and paths are stored differently in Design Framework II and CIF formats. The Design Framework II database does not store both the starting and ending points of a polygon because they are the same point. CIF requires that both points be stored individually. The largest Design Framework II polygon or path Export – CIF can process has 512 vertexes, which translates to 512 CIF vertexes.

Comprehensive Log indicates whether Export – CIF writes warning and error messages generated by SKILL function calls and PIPO dependencies to the PIPO log file during translation. Default: nil

Ignore Pcell Evaluation Failure lets you choose whether Export-CIF should ignore pcell evaluation failures or stop translation on encountering a pcell evaluation failure. If you do not select this option, Export-CIF stops translation when it encounters a pcell evaluation failure and a fatal message is displayed. Default: nil

Exporting a CIF File Using a Form

1. Select File – Export – CIF.
   The Virtuoso® CIF Out form appears.
2. Type the Design Framework II file to translate.
3. Type the output filename.
4. Specify any optional fields you want.
5. Click Apply or OK.
Export - CIF begins the conversion.

Exporting a CIF File from the Command Line

1. Create a template file by using the *Virtuoso*® *CIF Out* form.
   
   For information about how to create the template file, see the “Preparing a Template File” section in this chapter.

2. Enter the following command line:

   
   pipo cifout templateFile

Preparing a Template File and .cdsinit File

A collection of option values that you can load into the *Virtuoso*® *CIF Out* form can be defined in a template file or in the .cdsinit file. The required values in the CIF Out template file and the .cdsinit file are exactly the same. You can use either way to load the option values into the *Virtuoso*® *CIF Out* form. A template file can be loaded into the *Virtuoso*® *CIF Out* form in interactive mode or specified when you use the `pipo` command while the .cdsinit file is automatically loaded into *Virtuoso*® *CIF Out* form when layout is activated. You can create a template file in one of two ways:

- Enter values in the *Virtuoso*® *CIF Out*, *CIF Out Options*, and *CIF Out User-Defined Data* forms and click *Save* to save the option values to the file you specify as the template file.

- Create a text file. You can copy the sample template file cifOut.il in the samples/transUI directory and modify the file.

The required values in the CIF Out template file are the name of the input file and the name of the Design Framework II library. The other values are optional.

The following is a sample CIF Out template file:

```plaintext
cifOutKeys = list(nil
   'runDir "."
   'libName "" ;the Cadence source library
   'outFile ""
 ; output CIF specify
 ; 'scale 0.01
   'units "micron" ;("micron","millimeter","mil")
 ; select Cadence library data
 ; 'primaryCell ""
   'viewName "layout"
```


How Export – CIF Translates Data

There are important differences between CIF format and Design Framework II database format. To ensure successful translation of your Design Framework II data, familiarize yourself with the information in this section.

The major differences between Design Framework II format and CIF format are the following:

- Design Framework II format does not support zero-width paths. CIF does not support lines. Export – CIF translates lines to zero-width wires.

- The maximum number of points allowed in a CIF polygon is 1024. Export – CIF gives you a warning message when it encounters a polygon with more than 1024 points and does not translate the polygon.

- Export – CIF flattens magnified cells during translation.
Export — CIF translates symbolic devices such as pins, contacts, and transistors to cell instances. The following table shows how Export — CIF maps data types.

<table>
<thead>
<tr>
<th>Design Framework II data type</th>
<th>CIF data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell view</td>
<td>define symbol</td>
</tr>
<tr>
<td>polygon</td>
<td>polygon</td>
</tr>
<tr>
<td>path</td>
<td>wire</td>
</tr>
<tr>
<td>ellipse</td>
<td>polygon</td>
</tr>
<tr>
<td>circle</td>
<td>round flash</td>
</tr>
<tr>
<td>circle path</td>
<td>wire</td>
</tr>
<tr>
<td>arc path</td>
<td>wire</td>
</tr>
<tr>
<td>simple mosaic</td>
<td>call symbols</td>
</tr>
<tr>
<td>instance</td>
<td>call symbol</td>
</tr>
<tr>
<td>rectangle</td>
<td>box</td>
</tr>
<tr>
<td>donut</td>
<td>polygon</td>
</tr>
<tr>
<td>line</td>
<td>wire</td>
</tr>
<tr>
<td>symbolic device</td>
<td>geometric data</td>
</tr>
</tbody>
</table>

Export — CIF Output Files

In addition to a translated design data file, with input objects mapped into output objects, Export — CIF produces a log file and optional files you select.

Depending on what output files you specified in the run form or template file, Export — CIF produces one or more of the following files:

- A translated CIF file, with input objects mapped to output objects.
- A text version of the Design Framework II database file. When Export — CIF generates this file, it does not translate the file to CIF format.
- An error file (PIPO.LOG by default) which reports information, warning, and error messages that might have occurred during translation.
## Error File

The following is an example of an error file that Export – CIF produces. The Export – CIF parameter values used to produce this example are included at the end of the file.

```
*******************************************************************
PIPO Summary Report File
Exec Time = 02-Oct-1990   23:55:1
*******************************************************************
1. translating cellview (INV layout)
2. translating cellview (INV5 layout)
3. translating cellview (INV15 layout)
4. translating cellview (INV30 layout)
5. translating cellview (INV150 layout)
Top Cells In File
*******************************************************************
INV150 layout
List Hierarchy(#cellInst,#arrayInst)
*******************************************************************
INV150 layout
..INV15 layout(2)
...INV5 layout(3)
....INV layout(5)
Individual Cell Statistics
CellView Rectangles Polygons Paths Ellipses Labels Instances Arrays
*******************************************************************
INV/layout 11 3 0 0 4 0 0
INV/layout 0 0 0 0 0 5 0
INV/layout 0 0 0 0 0 3 0
INV/layout 2 2 0 0 0 2 0
INV/layout 2 0 0 0 4 5 0
Statistics Of Layers
Layer-# Rectangles Polygons Paths Ellipses Labels
*******************************************************************
1 1 5 0 0 4
2 0 0 0 0 4
3 4 0 0 0 0
4 1 0 0 0 0
5 1 0 0 0 0
6 8 0 0 0 0
Summary Of Objects
*******************************************************************
15 Rectangles
5 Polygons
0 Paths
0 Ellipses
8 Labels
15 Instances
0 Arrays
5 Cells
00:00:00 Elapsed time
00:00:02 CPU time
```
1132 Kbytes of memory

*** There were no error or warning messages ***

NORMAL EXIT ...

**Template File**

The template file contains

```plaintext
cifOutKeys = list(nil
   'runDir   "."'
   'libName  "myLib"
   'outFile  "1.cif"
   'scale   0.01
   'units   "micron"
   'primaryCell "inv150"
   'hierDepth 50
   'caseSensitivity "upper"
   'cellMapTable ""
   'layerTable ""
   'dotToPolygon nil
   'convertPin "drop"
   'pinTextMapTable nil
   'reportPrecision nil
   'checkPolygon  t
   'snapToGrid nil
   'errFile  "PIPO.LOG"
   'dataDump  ""
   'lineToZeroWire "wire"
   'magnifyToPolygon "ignore"
   'maxPolygonPoint 1024
   'userSkillFile ""
)
```

**ASCII Dump File**

If you select ASCII Dump in the Virtuoso® CIF Out form, Export – CIF produces an ASCII text image file of the Design Framework II database. If an ASCII dump file is generated, data is not translated. The following is a sample ASCII dump file:

```plaintext
Begin Library

Library Name : uclib, DB unit per user unit : 100, User Unit : micron
Begin Cell Definition

Cell Name : LABELS, View Name : layout
Text - Layer : 11 Text Type : 0 Origin : (-55,-130)
   String : \_'_{_}_[_]_:_<_>_._,_?_/_'_" Justification : 3 Angle : 0.000 Mirror : 0 Font : 6 Magnification : 1.000
Text - Layer : 11 Text Type : 0 Origin : (-55,-110)
   String : !_@_#_$_%_^_&_*_(_)___+_|_~_-_= Justification : 3 Angle : 0.000 Mirror : 0 Font : 6 Magnification : 1.000
Text - Layer : 11 Text Type : 0 Origin : (-55,-90)
   String : 0_1_2_3_4_5_6_7_8_9 Justification : 3 Angle : 0.000 Mirror : 0 Font : 6 Magnification : 1.000
```
Design Data Translator’s Reference
Translating CIF Files

Text - Layer : 11  Text Type : 0  Origin : (-55,-70)
   String : Qq_Rr_Ss_Tt_Uu_Vv_Ww_Xx_Yy_Zz  Justification : 3 Angle : 0.000 Mirror : 0
   Font : 6  Magnification : 1.000
Text - Layer : 11  Text Type : 0  Origin : (-55,-50)
   String : Ii_Jj_Kk_Ll_Mm_Nn_Oo_Pp  Justification : 3 Angle : 0.000 Mirror : 0
   Font : 6  Magnification : 1.000
Text - Layer : 11  Text Type : 0  Origin : (-55,-30)
   String : Aa_Bb_Cc_Dd_Ee_Ff_Gg_Hh  Justification : 3 Angle : 0.000 Mirror : 0
   Font : 6  Magnification : 1.000
Text - Layer : 11  Text Type : 0  Origin : (-55,-10)
   String : CADENCE_DESIGN_SYSTEMS_  Justification : 3 Angle : 0.000 Mirror : 0
   Font : 6  Magnification : 1.000
End Cell Definition
End Library
CDL Translators

This chapter describes the following:

- **Overview** on page 182
- **Using CDL In** on page 182
  - Running CDL In from the GUI on page 184
  - Running CDL In from the Command Line on page 189
  - How CDL In Translates Data on page 196
  - CDL In Output Files on page 203
  - CDL In SKILL Functions on page 204
- **Using CDL Out** on page 205
  - Running CDL Out from the GUI on page 209
  - Running CDL Out from the Command Line on page 219
  - How CDL Out Translates Data on page 225
  - CDL Out Formats on page 236
  - CDL Out Output Files on page 239
  - How to Customize CDL Netlist on page 240
  - CDL OUT SKILL Functions on page 241
Overview

Circuit Description Language (CDL) is a subset of Simulation Program Integrated Circuit Emphasis (SPICE) language. For more information about the CDL syntax, refer to Appendix B of the Assura Physical Verification Command Reference.

Using CDL In

CDL In takes a CDL netlist as input and prepares either a Virtuoso-Schematic or Virtuoso-Netlist view in the CDB database format.

To create a Virtuoso-Schematic view, you require any one of the following licenses:

- Virtuoso® Schematic Editor (34500)
- Virtuoso® Schematic Editor L (95100)
- Virtuoso® Schematic Editor XL (95115)
- Cadence Design FrameWork II (111)
- Virtuoso® Simulation Environment (206) license.

If one of the above mentioned licenses are not already checked out, the first available license will be checked out in the order mentioned above.
The following figure shows the inputs and outputs of CDL In.

You must specify a parameter file when you use the nino command to run CDL In. The nino command is the noninteractive, batch mode version of CDL In.
Running CDL In from the GUI

You can launch CDL In from any one of the following workbenches: icde, icdf, icms, layout, layoutPlus, and msfb. In the CIW, select File – Import – CDL. This displays the Virtuoso® CDL In form.

Parameter File

Specify the name of the parameter file. If the file is not located in the current working directory, specify the absolute path of the file. The parameter file is used to save values for the various fields in the form, thereby allowing the values used in one session to be reused in another session.
It is optional to use a parameter file if you run CDL In from the GUI. However, you must create and specify a parameter file when running CDL In from the command line. For more information about creating a parameter file, see Preparing a Parameter File on page 190.

**CDL Netlist File**

Specify the file you want to import. This is a mandatory field.

**Output Library**

Specify the name of the Design Framework II library where you want CDL In to place the translated cells. This is a mandatory field.

**Output View Type**

Specify the type of view you want to use for the translated cells in the Design Framework II database.

Valid Values: netlist and schematic
Default: schematic

**Output View Name**

Specify the name of the view you want to use for the translated cells in the Design Framework II database.

Default: schematic
**Top Cell**

Specify the top cell of the part of the hierarchy you want to translate. To translate only a specific cell and all of its descendants, specify that cell as the top cell.

For example, consider the following hierarchy:

```
        A
       / \  
      B   C
     / \  / \  
    F  D E  F
```

To translate all these cells, leave this field empty; CDL In translates the entire hierarchy. If you want to translate cells C, D, E, and F, but not A and B, type C in this field.

**Reference Library List**

Specify all the reference libraries or Process Design Kits (PDK) where the symbol views of master cells are present. Specify the list of library names by separating them with a space, tab, semicolon, or a comma.

**Device-Map File**

Specify the name of the device-map file. If the file is not located in the current working directory, specify the absolute path of the file. A device-map file primarily contains the mapping of primitive device names to user-defined cell names in any specified library. For more information about creating a device-map file, see [Preparing a Device-Map File](#) on page 192.
Global Node Expansion

Specify how CDL In should expand global nodes, defined in a `.global` statement in the CDL netlist, to extra terminals for each cell.

Valid Values:

- **Full**: Expands all global nodes to extra terminals for each cell.
- **No**: Does not expand global nodes to extra terminals for each cell.
- **Only Defined**: Expands only global nodes that are included in the cell subcircuit definition to extra terminals for each cell.

Default: **Full**

Case Sensitivity

Specify how CDL In should process the case of letters in cell names, node names, and instance names.

Valid Values:

- **Preserve**: Retains cell names, node names, and instance names without changing the case. For example, the name `Obj1` remains `Obj1`.
- **Upper**: Translates cell names, node names, and instance names to all uppercase. For example, the name `Obj1` becomes `OBJ1`.
- **Lower**: Translates cell names, node names, and instance names to all lowercase. For example, the name `Obj1` becomes `obj1`.

Default: **Preserve**

Schematic Generation Parameters

Click to invoke the Virtuoso® Schematic Generation form. For more information, see Connectivity to Schematic User Guide.
Overwrite Cells?

Specify whether to overwrite all, none or selected cellviews in the output library.

Valid Values: all, selected, and none
Default: all

To overwrite selected cells, specify the output library name and select selected from the Overwrite Cells? drop-down list box. CDL In automatically displays all the cells from the output library in the Cells in Output Library list. Using the Add >> and << Remove buttons, select the cells that you want to overwrite and the selected cells are displayed in the Cells to be Overwritten list.

You can limit the number of cells displayed in the Cells in Output Library list by specifying a regular expression in the Cells in Output Library text box. CDL In then displays only those cells that match the specified regular expression.
To run CDL In, perform the following steps by using the CDL In form:

1. In the CDL Netlist File field, type the name of the input CDL netlist file.
2. In the Output Library field, type the name of the output library.
3. In the Reference Library List field, type the names of the reference libraries.
4. Specify values for the optional fields you want.
5. Click Apply or OK.

   CDL In begins the translation.

⚠️ Important

You can set the following values in the .cdsenv file:

```plaintext
  cdlIn expandMOSMultiplicityFactor boolean nil nil
  cdlIn viewType string "schematic" nil
  cdlIn viewName string "schematic" nil
  cdlIn caseSensitivity string "Preserve" nil
  cdlIn hierarchy string "Full" nil
  cdlIn globalNodeExpansion string "Full" nil
  cdlIn overwriteCells string "all" nil
```

Running CDL In from the Command Line

1. Create a parameter file by using the CDL In form. You can also create the parameter file manually.

   See the section on “Preparing a Parameter File”.

2. Type the following at the command line:

   ```bash
   nino cdlin parameterFile
   ```
Preparing a Parameter File

A parameter file, written in SKILL language, is a list of parameters specified as name-value pairs. The various parameters map to the different fields in the GUI. The values of the parameters in the parameter file are used both to populate the fields in the GUI and also to save values from the GUI.

You can create a CDL In parameter file in the following ways:

- Use the Save command in the CDL In form to save the information currently entered in the form. The information is saved to the file you specify as the parameter file.
- Create a text file. You can copy the sample parameter file, cdlIn.il, in the samples/transUI directory and modify the file.

It is mandatory to specify the following parameters and their values in the CDL In parameter file: the CDL input file (cdlFile), the name of the output library (opusLib), and the name of the Design Framework II library (refLibList). The other parameters are optional.

Sample of a CDL In Parameter File

cdlInKeys = list(nil
'cdlFile            ""'
'opusLib            ""'
'primaryCell        ""'
'caseSensitivity    "preserve"
&viewName           "schematic"
&viewType           "schematic"
'refLibList          ""'
'globalNodeExpand   "full"
'schGenArgs         "+PLACE_ONLY -NOSQUARE -MIN_CROSSEOVERS -FAST_LABELS
                    +NOXTRSC -VERBOSE"
'schGenParamFile    "conn2sch_param"
'devMapFile         "/usr/1/home/bsingh/devMap"
'overwriteExistingCVs "selected"
)
Mapping Parameters with the CDL In GUI Options

**Parameter File Options**

- **cdlFile**
- **opusLib**
- **viewType**

**Valid Values:** schematic, netlist

- **viewName**
- **primaryCell**
- **refLibList**
- **devMapFile**
- **globalNodeExpand**
- **caseSensitivity**

**Valid Values:** preserve, upper, lower

**schGenArgs**

The mapping between the possible values of this parameter and the GUI fields in the *Virtuoso® Schematic Generation* form is given below:

- **+PLACE_ONLY** — Full Place and Route
- **-NOSQUARE** — Generate Square Schematics
- **-MIN_CROSSOVERS** — Minimize Crossovers
- **-FAST_LABELS** — Optimize Wire Label Locations
- **+NOXTRSCH** — Extract Schematics
- **-VERBOSE** — Verbose

**schGenParamFile**

This parameter has no equivalent GUI field. Using this parameter, you can specify a schematic generation parameter file that contains the schematic generation parameters and their values.
Preparing a Device-Map File

You can create a device-map file to map primitive devices to user-defined cells. For example, you can map the primitive device `pfet` to `pfetx` defined in a user-specified library. CDL In supports five primitive devices, `bjt`, `cap`, `diode`, `mos`, and `res`. You can specify the device-map filename in the `CDL In` form or in the parameter file by using the `devMapFile` option. The device-map file is case-insensitive.

It is essential to specify the primitive device name and the mapped device name. CDL In sequentially searches the mapped device name in the specified reference libraries and the output library. If the mapped device cellname is not found in any of the specified libraries, the primitive device is not mapped.

Syntax for Specifying Device Maps in a Device-Map File

```plaintext
devMap := <primitive_device_name> <mapped_device_name>
[ propMatch := list_of_prop_to_match ]
[ termMap := list_of_terminals_to_map ]
[ propMap := list_of_properties_to_map ]
[ addProp := list_of_additional_properties_to_map ]
```

**Note:** Use a blank space on either side of `:=`. 

In the syntax,

\[ \text{devMap} \]

Each device-map entry must begin with the `devMap` keyword. Although the `devMap` line can be followed by multiple `propMatch`, `termMap`, `propMap`, and `addProp` entries, only the first `propMatch`, `termMap`, `propMap`, and `addProp` entry is used; the subsequent entries are ignored. In the following example, `r` is mapped to `resVal` and not `resVal1`.

\[
\begin{align*}
\text{propMatch} & := r \text{ resVal w width l length} \\
\text{propMatch} & := r \text{ resVal1} \\
\text{propMatch} & := r \text{ resVal1} \\
\end{align*}
\]

- **primitive_device_name**: Name of the primitive device to be mapped.
- **mapped_device_name**: Name of the mapped device.
- **propMatch**: List of property name and property value pairs. Each specified property must be followed by the property value. The property names must exist in the specified primitive devices. Also, the values of each specified property must match else mapping will not be done for the device.
  
  An example of `propMatch` line is as follows:
  
  \[
  \text{propMatch} := \text{subType ND}
  \]

- **termMap**: List of primitive device terminal names and mapped device terminal name pairs. For example, if terminal list of the primitive device, `pfet`, is `D S G B` and the terminal list of the mapped device, `pfetx`, is `d s g b`, the `termMap` line should be defined as follows:
  
  \[
  \text{termMap} := D d S s G g B b
  \]

  The number of terminals in the `termMap` list must be even in number and in each mapping pair, the first terminal must exist in the primitive device and second in the mapped device. If a terminal does not exist in the corresponding `cellviews`, mapping is not done for that device.

  **Note**: The number of terminals in the primitive device must be equal to the number of terminals in the mapped device.
### propMap

List of properties if the properties of the mapped device are different from the primitive device. For example, if the primitive device, `pfet`, has a property `w` and the corresponding property in the mapped device, `pmos`, is `width`, then the propMap line is as follows:

```propMap := w width```

### addProp

List of additional properties and their values if the mapped device has user-defined string properties. Both property name and property value must be specified.

**Note:** The property value must be of `string` type. Also, CDL In does not validate the property value. The property is added as is to the instances of the mapped device.
Sample of a Device-Map File

This file contains two device map entries for resistor, resistor to res and resistor to res1.

```
devMap := resistor res
propMatch := r 20000
termMap := PLUS PLUS MINUS MINUS
propMap := subType type r resVal w width l length
addProp := model res

devMap := resistor res1
propMatch := r 10000
termMap := PLUS PLUS MINUS MINUS
propMap := subType type r resVal w width l length
addProp := model res
```

Now consider the following CDL instances:

```
R0 a b 20000 $w=10 $l=20
```

If the device-map file is specified, CDL In determines two mappings for resistor in the device-map file. After matching r with 20000 defined in the propMatch line, CDL In creates an instance of res. The w and l properties of resistor are mapped to width and length properties of res as defined in the propMap line. CDL In searches for res first in the reference libraries and then in the output library.

```
R0 a b 10000 $w=10 $l=20
```

For this CDL instance, CDL In creates an instance of res1 in the schematic after matching r with 10000 defined in the propMatch line.
How CDL In Translates Data

This section describes how CDL In translates the following:

- **Instances of Primitive devices**
- **Non-primitive devices**
- **Instance names**
- **Properties of Instances**
- **Global nodes**

How CDL In Translates Instances of Primitive Devices

CDL In translates primitive devices defined in subcircuit statements to instances. A primitive device is a basic physical component, such as a transistor or resistor. CDL In translates five types of primitive devices: mos, bipolar, resistor, capacitor, and diode.

CDL In uses the following information to translate instances of primitive devices:

<table>
<thead>
<tr>
<th>Instance Name</th>
<th>Master Cell Name</th>
<th>Terminal Order</th>
<th>Property Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>nmos</td>
<td>nfet</td>
<td>D, G, S, B</td>
<td>subtype, w, l</td>
</tr>
<tr>
<td>pmos</td>
<td>pfet</td>
<td>D, G, S, B</td>
<td>subtype, w, l</td>
</tr>
<tr>
<td>npn</td>
<td>npn</td>
<td>E, C, B</td>
<td>subtype, earea, w, l</td>
</tr>
<tr>
<td>pnp</td>
<td>pnp</td>
<td>E, C, B</td>
<td>subtype, earea, w, l</td>
</tr>
<tr>
<td>2-terminal resistor</td>
<td>resistor</td>
<td>PLUS, MINUS</td>
<td>subtype, r, w, l</td>
</tr>
<tr>
<td>3-terminal resistor</td>
<td>phyres</td>
<td>PLUS, MINUS, SUB</td>
<td>subtype, r, w, l</td>
</tr>
<tr>
<td>capacitor</td>
<td>capacitor</td>
<td>PLUS, MINUS</td>
<td>subtype, c</td>
</tr>
<tr>
<td>diode</td>
<td>diode</td>
<td>PLUS, MINUS</td>
<td>subtype, area, perim</td>
</tr>
</tbody>
</table>

Syntax for Instances of Primitive Devices

This section details the syntax for the following primitive devices:

- **Capacitor** on page 197
- **Diode** on page 198
Resistor on page 199
Transistor (BJT) on page 200
Transistor (JFET) on page 200
Transistor (MOSFET) on page 201

For more information and usage examples for creating their instances, see the chapter on LOGLVS in the *Dracula Reference*.

**Capacitor**

Cxxx npositive nminus {cap} {M=multiplier} {[$mname] / $.MODEL=mname} 
{[$SUB=substrate]}

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cxxx</td>
<td>The capacitor element. It must begin with the letter C.</td>
</tr>
<tr>
<td>npositive</td>
<td>The positive terminal node number/name.</td>
</tr>
<tr>
<td>multiplier</td>
<td>The multiplier device.</td>
</tr>
<tr>
<td>nminus</td>
<td>The negative terminal node number/name.</td>
</tr>
<tr>
<td>cap</td>
<td>The capacitance value. Dracula does not check the value if this specification is missing.</td>
</tr>
<tr>
<td>mname</td>
<td>Specifies the model name reference. The name must be one or two characters. If you want model/subtype checking, you must specify the mname in the CDL netlist and specify the ELEMENT CAP subtype option in your rules file.</td>
</tr>
<tr>
<td>substrate</td>
<td>Specifies substrate terminal or node name.</td>
</tr>
</tbody>
</table>

**Note:** The capacitor instance definition should be preceded with the following LOGLVS control statements:

* *.BIPOLAR *

* *.CAPVAL *

For more information about these control statements, see the chapter on LOGLVS in the *Dracula Reference*. 
Diode

Dxxx npositive nminus mname (AREA=area) (PJ=periphery) (M=multiplier) ($SUB=substrate)

Arguments

Dxxx The diode element name. It must begin with the letter D.
npositive The positive terminal node number/name.
nminus The negative terminal node number/name.
mname The model name reference. The name must be one or two characters. For model/subtype checking, you must specify the mname in the CDL netlist and specify the ELEMENT DIO subtype option in your rules file. You cannot swap npositive and nminus terminals.
area The diode area specification.
periphery The periphery of junction.
substrate The substrate terminal or node name.

Note: The diode instance definition should be preceded with the following LOGLVS control statements:

*.BIPOLAR

*.DIOAREA

*.DIOPERI

For more information about these control statements, see the chapter on LOGLVS in the Dracula Reference.
**Resistor**

```
Rxxx term1 term2 {res} {$SUB=substrate} {M=multiplier} {[$mname] /$.MODEL=mname} {$W=width} {$L=length}
```

**Arguments**

- **Rxxx** The resistor element name. It must begin with the letter R.
- **term1** The first terminal node number/name.
- **term2** The second terminal node number/mname.
- **res** The resistance value. Dracula does not check the value if this specification is missing.
- **substrate** The substrate terminal or node name.
- **multiplier** The multiplier device.
- **mname** The model name reference. The name must be one or two characters. For model/subtype checking, you must specify the mname in the CDL netlist and specify the ELEMENT RES command with a subtype option in your rules file.
- **width** The width of the resistor.
- **length** The length of the resistor.

**Note:** The resistor instance definition should be preceded with the following LOGLVS control statements:

* `.BIPOLAR`

* `.RESVAL`

For more information about these control statements, see the chapter on LOGLVS in the *Dracula Reference*. 
Design Data Translator's Reference
CDL Translators

Transistor (BJT)
Qxxx coll base emitter {nsub} mname {M=multiplier} {$EA=value} {$L=length}
{$W=width} {$SUB=substrate}

Arguments

Qxxx The BJT element name. It must begin with the letter Q.
coll The collector terminal node name.
base The base terminal node number/name.
emitter The emitter terminal node number/name.
nsub Substrate terminal node name or number.
mname The model name reference. The name must be one or two characters.
multiplier Specifies an integer. You must place multiplier in the description before you
specify any value.
value Specifies an emitter size, coded as a comment.
length The length of the BJT.
width The width of the BJT.
substrate The substrate terminal or node name.

Transistor (JFET)
Jxxx drain gate source mname {W=width L=length}

Arguments

Jxxx The JFET element name. It must begin with the letter J.
drain The drain terminal node number/name.
gate The gate terminal node number/name.
source The source terminal node number/name.
mname The model name reference. The name must be one or two characters.
width The width. The width is a real number. Dracula does not check the width if it
is not specified. W= is optional.
length The length. The length is a real number. Dracula does not check the length
if it is not specified. L= is optional.
**Transistor (MOSFET)**

Mxxx drain gate source bulk {/}mname {width {length}} {W=width} {L=length} {M=multiplier} {$LDD[type]} {$NONSWAP}

**Arguments**

- **Mxxx**: The MOSFET element name. It must begin with the letter M.
- **drain**: The drain terminal node number/name.
- **gate**: The gate terminal node number/name.
- **source**: The source terminal node number/name.
- **bulk**: The bulk terminal node number/name.
- **mname**: The model name reference. The name must be one or two characters.
- **width**: The optional width. Dracula does not check the width if this specification is missing.
- **length**: The optional length. Dracula does not check the length if this specification is missing.
- **multiplier**: The multiple device.
- **type**: The LDD designator, specified as a comment. Declares that the source and drain terminals have different characteristics and cannot be swapped.

**How CDL In Translates Instances of Non-Primitive Devices**

If a subcircuit definition contains an instance of a non-primitive device, CDL In searches for the symbol view of the master cell in the reference libraries.

**Example**

```
    .subckt nor2 Y A B
    M3 vdd A net3 vdd p w=10u l=2u
    M2 net3 B Y vdd p w=10u l=2u
    M1 Y B gnd gnd n w=5u l=2u
    M0 Y A gnd gnd n w=5u l=2u
    .ends nor2

    .subckt topcell net1 net6
    XI1 net3 net1 net2 nor2
    XI0 net6 net3 net2 nor2
    .ends topcell
```

CDL In will prepare netlist/schematic view of nor2. For instances XI0 and XI1, CDL In will search for nor2:symbol — first in the reference libraries and then in the output library.
How CDL In Translates Instance Names

CDL In reads only the first 50 characters of an instance name into the database. While reading the next instance, if it finds the first 50 characters to be identical to that of the existing one, it suffixes ‘-0’ to the second instance name. If the same situation happens for more instance names, CDL In suffixes ‘-1’, ‘-2’, and so on to the instance names.

How CDL In Translates Instance Properties

The master cell of a primitive device instance may have some component description properties (CDF) properties at the cell level. For example, analogLib/res/symbol has the CDF properties r and l among others. The default value of r is 1K, whereas l does not have a default value.

Now, consider the following resistor instance:

```
R0 a b 10k M=3 $[RX] $L=7u
```

R0 defines r as 10k, l as 7u, and subtype as RX. CDL In imports r and l as CDF properties. At the instance R0 level, the default value of r is overwritten by 10k, while l takes the value 7u.

If an instance property does not exist as a CDF property on the master cell, the property is imported as a user property on the netlist view or the schematic view.

⚠️ Caution

If you define a negative value for M, CDL In imports the values of other properties incorrectly.

How CDL In Translates Global Nodes

CDL In translates global nodes according to what you select for Global Node Expansion in the CDL In form. You define the global nodes with the .global statement in the CDL file.

Following is an example of how you define global nodes in a CDL file. In this example, if Global Node Expansion is full, CDL In creates two extra terminals (vcc and gnd) for the cell buffer. If Global Node Expansion is no, CDL In does not expand the global nodes (vcc and gnd) to extra terminals for the cell buffer. If Global Node Expansion is onlyDefined, CDL expands only the global node gnd to an extra terminal for the cell buffer because only node gnd is included in the cell content.

```
.global vcc:p gnd:g
.pin vcc,gnd
```
CDL In Output Files

Examples earlier in this chapter show what the CDL In output file in Design Framework II format looks like after translation. In addition to an output file, CDL In can produce the following three types of messages:

- **Error messages**, preceded by “ERROR (CDLIN-XXX)”, indicate serious, unrecoverable conditions. The converted file is incorrect. For example, if a cell instance is called but its master cell does not exist, an error message is written to the ni.err error file.

  **Note:** The exit status of CDL In when an error occurs is non-zero.

- **Warning messages**, preceded by “WARNING (CDLIN-XXX)”, indicate unexpected but recoverable conditions. The converted file is usable. For example, if a string is too long and has been truncated, a warning message is written to the ni.err error file.

  **Note:** When warning messages are generated, the exit status of CDL In is 0.

- **Information messages**, preceded by “INFO (CDLIN-XXX)”, indicate the status of a process that is running or the results of a completed process. Information messages are written to the file ni.log.
CDL In SKILL Functions

**cdlinGuiDisplay**

```python
cdlinGuiDisplay()
```

**Description**

Invokes the *CDL In* GUI form.

**Arguments**

None

**Value Returned**

None
Using CDL Out

Note: This CDL Out documentation is meant for the digital CDL netlist. For information about the analog CDL netlist (auCdl), refer to Appendix C, “auCdl Netlisting”, in the *Virtuoso Analog Design Environment User Guide*.

CDL Out translates a CDB Virtuoso-Schematic view into a CDL netlist. The following figure shows the inputs and outputs of CDL Out.

* You must specify an si.env file when you use the *si* command to run CDL Out. The *si* command is the noninteractive, batch mode version of CDL Out.
Preparing the Virtuoso-Schematic

The lowest levels of the schematic must contain primitives that CDL Out recognizes; for example, transistors, resistors, and capacitors. You must extract and save all levels of the schematic hierarchy before you netlist your design using CDL Out.

If you work on a circuit-level schematic and you create a circuit cell, which uses different parameter values for different instances of that cell, “parameter inheritance” is very important. For example, you might generate an inverter in your library and use it with 1X drive strength in one cell and with 2X drive strength in another. To ensure that the correct parameter values are passed to the appropriate instances, pass them through the hierarchy.

When you use hierarchical netlisting and parameter inheritance at the same time, special limitations apply. For information about these limitations, see the section How CDL Out Translates Parameters on page 227.

Creating an Instance Array

CDL Out lets you make an array from an instance. The specific instance is expanded with the name and property type. You can choose the number of times you want CDL Out to expand the instance.

<table>
<thead>
<tr>
<th>Name</th>
<th>Property type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDLMultiplier</td>
<td>integer</td>
</tr>
</tbody>
</table>

For example, if CDLMultiplier = 3 on instance MP8, the netlist appears as follows:

MN8.1 VDD IN OUT C21PLUS P
MN8.2 VDD IN OUT C21PLUS P
MN8.3 VDD IN OUT C21PLUS P

Creating a Black Box

Using black box properties, you can direct CDL Out to not netlist various blocks in a design. You add the black box properties directly to the property list of the symbol view.

The black box feature blocks out a particular cell by adding properties from the symbol view.
You can add the following properties to the symbol view of the instance master:

<table>
<thead>
<tr>
<th>Name</th>
<th>Property type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDLBlackBox</td>
<td>t/nil</td>
</tr>
<tr>
<td>CDLGenerateSubcircuitCard</td>
<td>t/nil</td>
</tr>
<tr>
<td>CDLPinList</td>
<td>string</td>
</tr>
<tr>
<td>CDLParameterNameList</td>
<td>string</td>
</tr>
<tr>
<td>CDLParameterValueList</td>
<td>string</td>
</tr>
</tbody>
</table>

A blackbox cell must also have a CDL view. To add a CDL view to a blackbox cell, copy its symbol view and save it as cdl.

**CDLBlackBox**

If CDLBlackBox does not exist and is equal to nil, CDL processing occurs as normal for that block. If the property exists and is equal to t, the CDL netlister checks the following properties:

<table>
<thead>
<tr>
<th>Name</th>
<th>Property type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDLGenerateSubcircuitCard</td>
<td></td>
</tr>
<tr>
<td>CDLPinList</td>
<td>I/O list of block</td>
</tr>
<tr>
<td>CDLParameterNameList</td>
<td>parameters of block</td>
</tr>
<tr>
<td>CDLParameterValueList</td>
<td>values of parameters of block</td>
</tr>
</tbody>
</table>

**CDLGenerateSubcircuitCard**

If you set the property CDLGenerateSubcircuitCard to nil, the CDL netlister does not generate a subcircuit. The CDL netlister orders the connections for all instances of a block according to the CDLPinList value. If you do not set the CDLPinList property, or set it to a null string, the CDL netlister orders the connections for all instances of a block alphabetically, using outputs, followed by bidirectionals, followed by inputs.
If you set the property `CDLGenerateSubcircuitCard` to `t`, the CDL netlister generates an empty subcircuit model card in the following form:

```
.SUBCKT AX PIN1 PIN2...PINN
.ENDS
```

**CDLPinList**

The property `CDLPinList` lists the order of connections for all instances in the block. If you do not set the `CDLPinList` property or set it to a null string, the CDL netlister makes the connections for all instances in the block alphabetically, using outputs, followed by bidirectionals, followed by inputs.

An example of the `CDLPinList` property format is as follows:

```
pin1 pin2 ...
```

The format is a string data type. You do not need to use double quotes. Separate each pin name by one or more blanks. If you do not need the `CDLPinList`, do not specify a value.

**Note:** For the definition of an instance, CDL Out looks for the `CDLPinList` order in the CDL view of the master cell.

**CDLParameterNameList**

The property `CDLParameterNameList` defines the parameters of the block with values specified by `CDLParameterValueList`. These values are overridden by explicit `Name=Value` assignments in each instance of the block. The netlister does not assign parameters to instances unless the parameters are defined in `CDLParameterNameList`. See the example in the section "Parameter-Value Inheritance and Schematic Levels".

**CDLParameterValueList**

The property `CDLParameterValueList` defines the value of the parameters listed by `CDLParameterNameList`. `CDLParameterNameList` is optional. If you do not list the values CDL Out uses default values.
Running CDL Out from the GUI

In the CIW of a Virtuoso workbench, select *File – Export – CDL*. This displays the *Virtuoso® CDL Out* form.
<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template File</td>
<td></td>
</tr>
<tr>
<td>Run in Background</td>
<td></td>
</tr>
<tr>
<td>Netlisting Mode</td>
<td></td>
</tr>
<tr>
<td>Top Cell Name</td>
<td></td>
</tr>
<tr>
<td>View Name</td>
<td></td>
</tr>
<tr>
<td>Library Name</td>
<td></td>
</tr>
<tr>
<td>Output File</td>
<td></td>
</tr>
<tr>
<td>Run Directory</td>
<td></td>
</tr>
<tr>
<td>Resistor Threshold Value</td>
<td>2000</td>
</tr>
<tr>
<td>Resistor Model Name</td>
<td></td>
</tr>
<tr>
<td>Equivalents</td>
<td></td>
</tr>
<tr>
<td>Include File</td>
<td></td>
</tr>
<tr>
<td>Check Resistors</td>
<td></td>
</tr>
<tr>
<td>Check Capacitors</td>
<td></td>
</tr>
<tr>
<td>Check Diodes</td>
<td></td>
</tr>
<tr>
<td>Scale</td>
<td></td>
</tr>
<tr>
<td>Shrink Factor for width(w) and length(l)</td>
<td></td>
</tr>
<tr>
<td>Check LDD</td>
<td></td>
</tr>
<tr>
<td>Display Pin Information</td>
<td></td>
</tr>
<tr>
<td>Map Bus Name From &lt;&gt; To []</td>
<td></td>
</tr>
<tr>
<td>Global Power Signals</td>
<td></td>
</tr>
<tr>
<td>Global Ground Signals</td>
<td></td>
</tr>
<tr>
<td>RenameList</td>
<td></td>
</tr>
<tr>
<td>Print Inherited Connections</td>
<td></td>
</tr>
</tbody>
</table>
Template File

Specify the name of the template file. If the file is not located in the current working directory, specify the absolute path of the file. The template file is used to save values for the various fields in the form, thereby allowing the values used in one session to be reused in another session.

Load  Specify a template file in the Template File field and click Load. The values of fields in the form are updated with the values from the template file.

Save  After updating one or more fields, click Save. The values of the fields are saved to the template file specified in the Template File field.

Run In Background

Select to cause CDL Out to run in background mode. This option is selected by default for the Netlisting Mode digital. When this option is selected, you can edit a design while CDL Out runs. This mode is useful when you run CDL Out on large designs.

Default: on

Netlisting Mode

Select either of the two netlisting modes. The supported modes are analog and digital.

Valid Values:

analog  Selects auCdl.
digital  Selects CDL Out.

Default: digital

Note: If you select analog, the Run In Background option gets disabled because the analog CDL netlister (auCdl) runs only in the background.
Library Browser

Click to launch the Library Browser window. Specify a library name, top cell name, and view name. These values are used to populate the next three fields in the CDL Out form.

Top Cell Name

Specify the name of the cell at the top of the hierarchy that you want to netlist. For example, assume the cell hierarchy is as follows:

```
    A
   / \  
  B   C
   \ /  
   F   D
    \  
     E
```

If you want to translate all cells to the output file, enter A in this field. If you want to translate just cells C, D, E, and F, but not A and B, type C in this field.

View Name

Specify the view name of the design you want to translate.

Default: schematic

Library Name

Specify the name of the library that contains the view you want to translate from CDB Virtuoso-Schematic view into a CDL netlist.

Output File

Specify the name of the output CDL netlist file. CDL Out creates the output file in the run directory you specify in the next field.
Default: **netlist**

**Run Directory**

Specify the path of the directory in which you want CDL Out to place the intermediate files.

Default: current working directory (.)

**Resistor Threshold Value**

Specify the threshold value at which the resistors should short. You can change the default value by declaring a variable in the .cdsinit file. In a situation where you require a **Resistor Threshold Value** of 500 as the default, enter the following line in the .cdsinit file:

```
CdlDefaultShortResi = 500.00
```

Default: **2000 (ohms)**

**Resistor Model Name**

Specify the resistor model that you want to check.

**Equivalents**

List any equivalent items. Everything you type in this field appears in the netlist following the keyword *.EQUIV. For example, if you type a=b c=d in the **Equivalents** field, the line in the netlist shows:

```
*.EQUIV a=b c=d
```

**Include File**

List the file names to be included during the run. Each file name appears on a separate line following the keyword .INCLUDE. For example, if you enter the two file names abc def in the **Include File** field, the netlist shows:

```
.INCLUDE abc
.INCLUDE def
```
Check Resistors

Specify whether CDL Out should print a CDL control command related to resistors in the output CDL netlist file.

Valid Values:

- **value**: Prints the CDL control command *.RESVAL in the output CDL netlist file.
- **size**: Prints the CDL control command *.RESIZE in the output CDL netlist file.
- **none**: Prints neither *.RESVAL nor *.RESIZE in the output CDL netlist file.

Default: **value**

Check Capacitors

Specify whether CDL Out should print a CDL control command related to capacitors in the output CDL netlist file.

Valid Values:

- **value**: Prints the CDL control command *.CAPVAL in the output CDL netlist file.
- **area**: Prints the CDL control command *.CAPAREA in the output CDL netlist file.
- **perimeter**: Prints the CDL control command *.CAPPERI in the output CDL netlist file.
- **both**: Prints both the CDL control commands *.CAPAREA and *.CAPPERI in the output CDL netlist file.
- **none**: Prints neither *.CAPAREA nor *.CAPPERI in the output CDL netlist file.

Default: **value**
Check Diodes

Specify whether CDL Out should print a CDL control command related to diodes in the output CDL netlist file.

Valid Values:

- **area**: Prints the CDL control command *.DIOAREA in the output CDL netlist file.
- **perimeter**: Prints the CDL control command *.DIOPERI in the output CDL netlist file.
- **both**: Prints both the CDL control commands *.DIOAREA and *.DIOPERI in the output CDL netlist file.
- **topology**: Checks diode topology.
- **none**: Prints neither *.DIOAREA nor *.DIOPERI in the output CDL netlist file.

Default: **both**

During translation, the use of Check Resistors, Check Capacitors, and Check Diodes options determines which of the following statements are written to the output netlist:

- *.BIPOLAR
- *.RESI
- *.RESSIZE
- *.RESVAL
- *.CAPA
- *.CAPVAL
- *.CAPAREA
- *.CAPPERI
- *.DIODE
- *.DIOAREA
- *.DIOPERI
These statements appear in different combinations depending on the values selected for the *Check Resistors*, *Check Capacitors*, and *Check Diodes* options. This is shown in the following table.

<table>
<thead>
<tr>
<th>Check Resistors</th>
<th>Check Capacitors</th>
<th>Check Diodes</th>
<th>Output Netlist Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>none</td>
<td>none</td>
<td>None</td>
</tr>
<tr>
<td>size</td>
<td>none</td>
<td>none</td>
<td>*.BIPOLAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.RESSIZE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.CAPA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.DIODE</td>
</tr>
<tr>
<td>value</td>
<td>none</td>
<td>none</td>
<td>*.BIPOLAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.RESI = 2000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.RESVAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.CAPA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.DIODE</td>
</tr>
<tr>
<td>none</td>
<td>value</td>
<td>none</td>
<td>*.BIPOLAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.CAPVAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.DIODE</td>
</tr>
<tr>
<td>none</td>
<td>area/perimeter</td>
<td>none</td>
<td>*.BIPOLAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.CAPAREA/</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.CAPPERI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.DIODE</td>
</tr>
<tr>
<td>none</td>
<td>none</td>
<td>area/perimeter</td>
<td>*.BIPOLAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.CAPA</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.DIOAREA/</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.DIOPERI</td>
</tr>
<tr>
<td>none</td>
<td>none</td>
<td>topology</td>
<td>*.BIPOLAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.CAPA</td>
</tr>
<tr>
<td>value</td>
<td>none</td>
<td>topology</td>
<td>*.BIPOLAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.RESI = 2000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.RESVAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.CAPA</td>
</tr>
<tr>
<td>size</td>
<td>none</td>
<td>topology</td>
<td>*.BIPOLAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.RESI = 2000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.RESSIZE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*.CAPA</td>
</tr>
</tbody>
</table>
Scale

Specify the scale to be used in the generated netlist by CDL Out.

Valid Values:

- **meter**: Prints `*.SCALE METER` in the netlist.
- **micron**: Prints `*.SCALE MICRON` in the netlist.
- **none**: Does not print `*.SCALE` in the netlist.

Default: **meter**

Shrink Factor for width(w) and length(l)

Specify the percentage by which you want the design to shrink. This field takes a real number. For example, if the transistor width is 10 and length is 1 and you require to shrink the transistor by 80%, then in the generated netlist, the width is 2 and the length is 0.2.

Default: 0

Check LDD

Select to enable CDL Out to print the statement `*.LDD` in the netlist.

Default: off
Display Pin Information

Select to enable CDL Out to display the pin type information.

Default: on

Map Bus Name From <> To []

Sets the mapping of bus pins to []. Once set, the netlister will generate the netlist with that bus mapping only. To change the mapping, you have to exit and restart the application and set the required bus mapping again.

Default: false

Global Power Signals

Specify a list of global power signals. The global power signals specified here will have a designator :P attached to them in the netlist.

Global Ground Signals

Specify a list of global ground signals. The global ground signals specified here will have a designator :G attached to them in the netlist.

Note: If the fields are not null, then signals specified in the list will be assumed to be final global signals. No attempt will be made to search in CDB and then to append/prepend global signals found in CDB (if different).

Renetlist

Sets the value of the parameter simReNetlistAll in the si.env file to t.

Default: off
Print Inherited Connections

Prints the constructs associated with inherited connections in the output CDL netlist file. This option is the GUI equivalent of the variable \texttt{simPrintInhConnAttributes}, which can be set in the \texttt{.simrc} file.

Default: off

To run CDL Out, perform the following steps by using the \textit{CDL Out} form:

1. In the \textit{Top Cell Name} field, type the name of the top cell.
2. In the \textit{Library Name} field, type the name of the library containing the view you want to netlist.
3. Specify values for the optional fields you want.
4. Click \textit{Apply} or \textit{OK}.

   CDL Out begins the translation.

Running CDL Out from the Command Line

1. Prepare the \texttt{si.env} file. Among other information, the \texttt{si.env} file carries the name of the design to be netlisted.

   For more information, see the section Preparing an \texttt{si.env} File on page 221.

2. Type the following at the command line:

   \texttt{si -batch -command netlist}

Automatic Netlist Inclusion

This feature allows you to include the CDL netlist of the blocks, which do not have a schematic view, in the final top level CDL netlist. During CDL Out, it is helpful if some of the blocks in the design do not have a schematic view but their CDL netlist is available. For such blocks, perform the following steps:

1. Create a blank schematic view.
2. In the schematic view, create dummy pins. The pin names should match with the pins present in the corresponding symbol view.
3. Create the string property `nlAction` on the schematic and symbol view and set its value to `stop`.

4. Create a boolean property `CDL_INCLUDE_FILE` on the schematic view and set its value to `TRUE`.

5. Copy the CDL netlist file `cdlNetlist` of this cell in the schematic view directory.

Now, if you run CDL out on the top level cell, then it will not create the `.subckt` in the netlist for the cellviews which have property `nlAction` set as `stop`. For such cellviews, if the `CDL_INCLUDE_FILE` property is set to `true`, then its netlist file `cdlNetlist` is copied as:

```
./includeFiles/libName_cellName_viewName
```

All these files are included at the bottom of the netlist with the help of `INCLUDE` statement.

**Preparing a Template File**

A template file is a collection of file names and option values that you can load into the `Virtuoso® CDL Out` form. You can create a CDL Out template file in one of two ways:

- Enter values in the `Virtuoso® CDL Out` form and click `Save` to save the option values to the file you specify as the template file.

- Create a text file. You can copy the sample template file `cdlOut.il` in the `samples/transUI` directory and modify the file.

The required values in the CDL Out template file are the name of the Design Framework II library containing the top level cellview and the top cell name to translate. The other values are optional.

The following is the sample CDL Out template file Cadence supplies in `samples/transUI/CDLOut.il`:

```plaintext
cdlOutKeys = list(nil
   'simLibName    "opus"
   'simCellName   "latch.cdl"
   'simViewName   "schematic"
   'hnlNetlistFileName    "netlist"
   'simRunDir     "."
   'shortRES      2000.000000
   'resistorCheck "none"
   'capacitorCheck "none"
   'diodeCheck   '('t t)
   'displayPinInfo 'nil
)
```

July 2007 220 Product Version 5.1.41
Pin Order for Top Cells

CDL Out, by default, prints the subcircuit pins in the following order:

Output   Bidirectional   Input

The pin order shown above is applicable for all the subcircuits in the design including the top cell. You can override the default pin order for the top cell by defining the SKILL variable cdlTopCellPinOrder in the .simrc file. The cdlTopCellPinOrder variable must have a string that fulfills the following format requirements:

- The string must consist of O, B, I, and / characters.
- The characters O, B, and I must appear once in a string.
- The characters O, B, and I can appear multiple number of times.
- The character / can be used between other characters.

Example

Assume that you want to specify the pin order for top cells as: input pins, followed by bidirectional pins, followed by /, and then by bidirectional pins and output pins. To do this, set the cdlTopCellPinOrder variable in the .simrc file as follows:

```plaintext
cdlTopCellPinOrder = "IB/BO"
```

Preparing an si.env File

To run CDL Out from command line, you must prepare an si.env file. The si.env file contains few mandatory and few optional variables. If you run CDL Out from the CIW, the CDL Out creates an si.env file automatically in the run directory. For more information about the si.env file, see the Open Simulation System (OSS) Reference.

Caution

*The .simrc control file in your home directory overwrites the si.env file in the run directory.*

Sample of an si.env File

```plaintext
simLibName = "opus"
simCellName = "latch.cdl"
simViewName = "schematic"
simReNetlistAll = "nil"
hnlNetlistFileName = "netlist"
simRunDir = "/cds/1.0/test/translator/cdlout/paramCase/"
```
simSimulator = "cdl"
simViewList = '("cdl" "schematic" "gate.sch" "symbol")
simPrintInhConnAttributes = ´nil
simNetNamePrefix = N
simInstNamePrefix = X
simModelNamePrefix = M
hnlMaxLineLength = 79

preserveALL = t
preserveRES = t [XX]
shortRES = 250.000000
checkRESSIZE = t
preserveCAP = t
checkCAPAREA = t
preserveDIO = t
checkDIOAREA = t
retainBusses = t
CDLUsePortOrderForPinList = ´nil

A few of these variables are described below.

**preserveAll**
If the property **preserveAll** is set to t, instances of resistors, capacitors, and diodes in the input schematic are netlisted. However, if **preserveAll** is set to nil, CDL Out does not print instance definitions of resistors, capacitors, and diodes in the output CDL netlist.

Default: nil

**preserveRES**
If you set the property **preserveRES** to t, resistors are preserved for checking in LVS. You can define the variables **shortRES** and **checkRESSIZE** if **preserveRES** = t. Using the optional variable [XX], you can specify a model name that preserves only the specified type of resistor.

Default: nil

**shortRES**
The value you set for **shortRES** defines the threshold of the resistors that are shorted.

**checkRESSIZE**
The value you set for **checkRESSIZE** is the resistor size. If you set a value for **checkRESSIZE**, LVS will check the resistor size when you run it.

Default: nil

**preserveCAP**
If you set the property **preserveCAP** to t, CDL Out preserves capacitors for checking in LVS. You can define the variable **checkCAPAREA** if **preserveCAP** = t.
Default: nil

**checkCAPAREA**
If you set the property `checkCAPAREA` to `t`, LVS will check the capacitor area when you run it.

Default: nil

**preserveDIO**
If you set the property `preserveDIO` to `t`, CDL Out preserves the diodes for checking in LVS. You can define the variable `checkDIOAREA` if `preserveDIO = t`.

Default: nil

**checkDIOAREA**
If you set the property `checkDIOAREA` to `t`, LVS will check the diode area when you run it.

Default: nil

**retainBusses**
If you set the property `retainBusses` to `t`, busses are printed in the same way in the output CDL netlist file as they are in the input schematic. When set to `nil`, CDL Out expands the busses in the schematic to individual scalars.

The following example shows how the display of the busses differs in the output CDL netlist file with the use of this property:

When `retainBusses = t`,

```
BUS<0:3>
```

When `retainBusses = nil`,

```
BUS<0> BUS<1> BUS<2> BUS<3>
```

Default: nil

**CDLUsePortOrderForPinList**
You can set the SKILL variable `CDLUsePortOrderForPinList` in `si.env` or `simrc`. If you set `CDLUsePortOrderForPinList` to `t`, then the netlist pin order is determined from the `portOrder` property present on the schematic view of a cell, otherwise, the `portOrder` property is ignored.

If the `CDLPinList` property is set on a cellview, then for the corresponding subcircuit in the CDL netlist, pin ordering will be done by the order in which `CDLPinList` has been specified. This overrides the `portOrder` property.
If the `CDLUsePortOrderForPinList` is not set to `t` and the `CDLPinList` property has not been specified on a cellview, then the pin order is printed as is available in the database: the output pin first, followed by the input-output and the input pins.

For example, consider the portOrder of the cellview `testLib/cell/schematic` to be ("A" "H" "Y") and assume that the property `CDLUsePortOrderForPinList` is either not set or set to `nil`. In this case, the list of terminals as retrieved from the database would be ("H" "A" "Y"), with the directions "inputOutput", "input", and "output" respectively. The pin list in the output CDL netlist would be "Y H A", with the output terminal appearing first.

The `CDLUsePortOrderForPinList` option is applicable to all the cells, including blackboxes.

Default: `nil`
How CDL Out Translates Data

There are important differences between CDL format and the Virtuoso-Schematic view. CDL Out generates a netlist hierarchy that duplicates the hierarchy of your design. Each cell in the schematic becomes a separate subcircuit in the netlist. The hierarchical netlister automatically prefixes each instance name with the proper character for its element type; for example, “M” for MOSFET and “R” for resistor. This minimizes mapping and name translation.

CDL Out names instances and nets differently than a flat netlister. A flat netlister maps all the names to unique names. This avoids naming conflicts if you use identical names for instances in different cells of your schematic.

Unlike a flat netlister, CDL Out maps only illegal names to new names in the hierarchical netlist. CDL Out considers a name illegal if it contains illegal characters or is more than 60 characters. CDL Out searches first for illegal characters, which it maps, as shown in the following table.

<table>
<thead>
<tr>
<th>First character</th>
<th>Net</th>
<th>Instance</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ ( ) [ ] &lt;&gt; @ ! , $</td>
<td>N</td>
<td>X</td>
<td>M</td>
</tr>
<tr>
<td>0 to 9</td>
<td>N0 to N9</td>
<td>X0 to X9</td>
<td>M0 to M9</td>
</tr>
<tr>
<td>A to Z</td>
<td>X_A to X_Z</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a to z</td>
<td>X_a to X_z</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For example, if #333 is a net name, CDL Out maps it to N333. If #333 is an instance name, CDL Out maps it to X333. And if #333 is a model name, CDL Out maps it to M333.

CDL Out then strips the following illegal characters from the name:

- period .
- comma ,
- open parentheses (
- close parentheses )
- open bracket [
- close bracket ]
- dollar sign $
Caution

To avoid conflicting instance names, if a name uses less-than and greater-than signs (< and >), CDL Out strips the greater-than sign from the name and replaces the less-than sign with an underscore. For example, the instance named x1<0> would be mapped to x1_0.

If a name has more than 60 characters, CDL Out maps it to a unique number preceded by one of the following letters:

N for net names
X for instance names
CDL_ for macro names

CDL Out strips the following illegal characters from the net names:

period .
comma ,
open parentheses ( 
close parentheses )
open bracket [ 
close bracket ]
This section describes how CDL Out translates the following:

- **Parameters**
- **Inherited Connections**
- **Primitive Components**
- **Global Signals**

### How CDL Out Translates Parameters

This section explains the rules for using different types of parameters and parameter inheritance.

Circuit-level simulations frequently involve element and model parameters. When combining parameter inheritance and hierarchical netlisting, Design Framework II parameter inheritance mechanisms are more general than those that CDL supports. The graphic display of the schematic might show different parameter values than those printed in the netlist.

To avoid this problem, when you create your schematics follow the guidelines in this section for assigned parameters, inherited parameters, and inheritance of inherited parameters.

#### Fixed-Value Inheritance

Most parameters have a fixed value for simple circuits. For example, if you want the instance `m10` transistor in your schematic to have a width of 20 microns, add

```
w = 20u
```

to the property list of the instance representing the `m10` transistor.

In more complex circuits, parameters can be inherited through the schematic hierarchy. Instead of assigning a fixed value to a parameter, use an `nlpExpr` value enclosed in brackets and separated by colons.

Now if you set the `w` parameter of the `m10` transistor to the following:
the \( w \) property is an \texttt{nlpExpr} type, indicating to CDL Out that \( w \) is an inherited parameter. CDL Out assigns the \( wn \) value to \( w \), if it exists, or assigns the default value of \( 5u \) if \( w \) does not exist.

The arguments in the parameter assignment are as follows:

- \texttt{wn} Name of the property whose value to assign to \( w \).
- \% Format argument that causes CDL Out to set the first argument to the default value specified in the third argument, if the first argument doesn’t have a specified value.
- \( 5u \) Default value for all unspecified elements of this property.

Separate these arguments with colons (:).

The following table shows the parameters to use for sizing the width of a pmos device that is used in building a hierarchy. CDL Out requires the parameters \( w \) and \( l \) at the bottom level. The names at higher levels can be different as long as they are legal names.

<table>
<thead>
<tr>
<th>Level</th>
<th>Symbol</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (device)</td>
<td>[@w]</td>
<td>( w=[@pw1] )</td>
</tr>
<tr>
<td>2 (gate)</td>
<td>[@pw1]</td>
<td>( pw1=[@pw2] )</td>
</tr>
<tr>
<td>3 (block)</td>
<td>[@pw2]</td>
<td>( pw2=[@pw3] )</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>N (block)</td>
<td>[@pw(N-1)]</td>
<td>( pw(N-1)=VALUE )</td>
</tr>
</tbody>
</table>

For further information about inherited property syntax, refer to the \textit{Open Simulation System Reference Manual}.

\textbf{Assigned Parameters}

You can also assign default values to parameters. If you want all transistors to have a length of 1.2 microns, add the \( l = 1.2u \) property to the CDL view of the transistor master. CDL Out uses this default length unless you add the \( l \) property to a particular transistor to override it for that instance.
To assign default properties for netlisting, add them to the CDL view of the element. To assign default parameters for graphic display, add them to the symbol view of the element.

CDL Out prints the assigned parameters for instances, nets, and macro elements on the corresponding lines in the netlist.

Any user property defined on an instance in the schematic is printed along with its value in the instance definition of that subcircuit in the output CDL netlist file. This is true for instances of both primitive and non-primitive devices.

If you do not want to print certain properties, you can list the properties in the cdlOutSkipInstanceProps SKILL variable in the .simrc file. For example, including the following line in the .simrc file does not print the partName and vendorName properties in the output CDL netlist file.

cdlOutSkipInstanceProps=list(‘partName’ ‘vendorName’)
Parameter-Value Inheritance and Schematic Levels

If you want an element to inherit a parameter value from another element, that parameter must come from a higher schematic level.

For example, if you have an `inv` inverter in your schematic that contains transistor `w10`, and you place an instance of that inverter named `inv1`, transistor `w10` in `inv1` inherits the width specified in the property list of the original `inv` inverter. If you want transistor `w10` of inverter `inv1` to have a width of 25u, you must add the following property to the `inv` property list:

```
wn=25u
```

The hierarchical netlist reflects this specification as follows:

```
xinv1 node1 node2 ... inv ... wn = 25u ...
```

The width assignment in the `inv` property list overrides the default value for width on the `.subckt` definition of the `inv1` schematic. You are not limited to making these assignments within individual instances of `inv`. You can make the `wn` assignment at any level above where `inv1` is used because parameter assignments apply down through the hierarchy.

To assign a default value to a parameter that another element inherits, you must add it to the `nlpExpr` of that parameter or it is ignored by CDL Out.

Inheritance of Inherited Parameters

You can assign the value of an inherited parameter to the value of another inherited parameter if you make the assignments at different levels and if the second assignment is at a higher level in the schematic. For example, you cannot assign `w = [@wn:%:5u]` on the instance representing transistor `m10`, and also assign `wn = 25u` on the same instance, because they are on the same level.

Refer to the Dracula Reference Manual for information about valid parameters for CDL Out.

How CDL Out Translates Inherited Connections

When CDL Out netlists a design that contains inherited connections, it creates dummy ports (or pseudo ports) to maintain connectivity across modules and their instantiations. These dummy ports introduce unwanted nets in the hierarchy and also result in the loss of data when design information is shared across various EDA tools in the flow.

You can turn off the creation of the dummy ports by using the CDL Out GUI check box option Print Inherited Connections. The corresponding switch in the `si.env` file is `simPrintInhConnAttributes`. It is a boolean variable with the default value `nil`. This
means that in the default state CDL Out will continue to create the dummy ports for inherited connections in a design. You can turn off the creation of the dummy ports by setting this variable to t. You can set this environment variable in the .simrc file.
The two statements that are printed in the output CDL netlist when simPrintInhConnAttributes is set to t are *.NETEXPR and $netSet.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>*.NETEXPR</td>
<td>*.NETEXPR prop_name netName default_signal</td>
<td>*.NETEXPR vdd vdd!</td>
</tr>
<tr>
<td>$netSet</td>
<td>$netSet prop_name = value</td>
<td>$netSet vdd = VDD!</td>
</tr>
</tbody>
</table>

**Support for Inheritance and Transfer of Supply Sensitivity Data**

CDL Out supports the SUPPLYSENSITIVITY constructs that enable you to combine logical and physical data in the flow. The use of these constructs ensures that there is no loss of supply sensitivity data when information is passed across various EDA tools in the design flow from the design entry to the verification stage.

The SUPPLYSENSITIVITY constructs include *.SUPPLYSENSITIVITY and *.GROUNDSENSITIVITY. These constructs have the following syntax:

**Syntax of *.SUPPLYSENSITIVITY**

*.*SUPPLYSENSITIVITY netName value

**Syntax of *.GROUNDSENSITIVITY**

*.*GROUNDSENSITIVITY netName value

**Note:** These property values are printed in the netlist corresponding to the nets to which these terminals are connected.

Since CDL Out does not distinguish between terminals and the nets that connect to these terminals, CDL Out uses the *.PINMAP statement that defines the mapping between a net and the cell terminal to which it is connected.

**Syntax of *.PINMAP**

*.*PINMAP net1: cell_terminal1 net2: cell_terminal 2...

**Example**

*.*PINMAP GND1:gnd! VDD1:vdd!
How CDL Out Translates Instances of Primitive Devices

This section describes how you must define an instance of a primitive device in the schematic so that CDL Out netlists the instance. CDL Out recognizes instances of only those cells that have both a symbol view and a CDL view.

You can store default values of parameters in the symbol view so that they are displayed in the schematic. In the CDL view, store the default values so that they are printed in the output CDL netlist.

To create CDL views, use the Symbol/Simulation Library Generator (S/SLG) described in the Virtuoso Schematic Composer User Guide.

Given below is a complete list of formatting functions (hnlCDLFormatInst) and parameter list variables (hnlCDLParamList) for all the primitive cells supported by CDL Out:

<table>
<thead>
<tr>
<th>Component</th>
<th>hnlCDLFormatInst</th>
<th>hnlCDLParamList</th>
</tr>
</thead>
<tbody>
<tr>
<td>BJT</td>
<td>hnlCDLPrintBJTElement</td>
<td>hnlCDLBJTParamList</td>
</tr>
<tr>
<td>Capacitor</td>
<td>hnlCDLPrintCapacitorElement</td>
<td>hnlCDLCapacitorParamL</td>
</tr>
<tr>
<td>Diode</td>
<td>hnlCDLPrintDiodeElement</td>
<td>hnlCDLDiodeParamList</td>
</tr>
<tr>
<td>ICIsrc</td>
<td>hnlCDLPrintICIsrcElement</td>
<td>hnlCDLICIsrcParamList</td>
</tr>
<tr>
<td>ICVsrc</td>
<td>hnlCDLPrintICVsrcElement</td>
<td>hnlCDLICVsrcParamList</td>
</tr>
<tr>
<td>Inductor</td>
<td>hnlCDLPrintInductorElement</td>
<td>hnlCDLInductorParamList</td>
</tr>
<tr>
<td>Isrc</td>
<td>hnlCDLPrintIsrcElement</td>
<td>hnlCDLIsrcParamList</td>
</tr>
<tr>
<td>Jfet</td>
<td>hnlCDLPrintJfetElement</td>
<td>hnlCDLJfetParamList</td>
</tr>
<tr>
<td>MOSfet</td>
<td>hnlCDLPrintMOSfetElement</td>
<td>hnlCDLMOSfetParamList</td>
</tr>
<tr>
<td>NMOSfet</td>
<td>hnlCDLPrintNMOSfetElement</td>
<td>hnlCDLMOSfetParamList</td>
</tr>
<tr>
<td>PMOSfet</td>
<td>hnlCDLPrintPMOSfetElement</td>
<td>hnlCDLMOSfetParamList</td>
</tr>
<tr>
<td>Resistor</td>
<td>hnlCDLPrintResistorElement</td>
<td>hnlCDLResistorParamList</td>
</tr>
<tr>
<td>Tline</td>
<td>hnlCDLPrintTlineElement</td>
<td>hnlCDLTlineParamList</td>
</tr>
<tr>
<td>VCIsrc</td>
<td>hnlCDLPrintVCIsrcElement</td>
<td>hnlCDLVCIsrcParamList</td>
</tr>
<tr>
<td>VCVsrc</td>
<td>hnlCDLPrintVCVsrcElement</td>
<td>hnlCDLVCVsrcParamList</td>
</tr>
<tr>
<td>Vsrc</td>
<td>hnlCDLPrintVsrcElement</td>
<td>hnlCDLVsrcParamList</td>
</tr>
</tbody>
</table>

For details of the above mentioned SKILL functions, see CDL OUT SKILL Functions.
**hnlCDLFormatInst**  
Specifies what formatting procedure to call to print the line of that type of element. For example, the value of this property for the nmos components in cdslib is:

```
hnlCDLFormatInst="hnlCDLPrintNMOSfetElement()"
```

**hnlCDLParamList**  
Specifies which parameters can be inherited. You must set the value of this parameter to the name of a SKILL variable that is itself a list of the parameters that can be inherited. Any parameter not in this list cannot inherit a value and can only be assigned a fixed value. For example, the value of this property for the nmos components in cdslib is:

```
hnlCDLParamList="hnlCDLMOSfetParamList"
```

**hnlCDLElementSubType**  
Specifies the element type. For example, the value of this property for the nmos components in cdslib is n.

```
hnlCDLElementSubType="n"
```

The following example shows part of a file that creates CDL views with values of these three properties for different components.

```
lmDefViewProp(cap cdl
  hnlCDLParamList = "hnlCDLCapParamList"
  hnlCDLFormatInst = "hnlCDLPrintCapElement()"
  hnlCDLElementSubType = "c1"
)
```

```
lmDefViewProp(capacitor cdl
  hnlCDLParamList = "hnlCDLCapacitorParamList"
  hnlCDLFormatInst = "hnlCDLPrintCapacitorElement()"
  hnlCDLElementSubType = "c2"
)
```

```
lmDefViewProp(diode cdl
  hnlCDLParamList = "hnlCDLDiodeParamList"
  hnlCDLFormatInst = "hnlCDLPrintDiodeElement()"
  hnlCDLElementSubType = "di"
)
```

The complete parameter list that can be present for each component is given in the CDL Out Formats section.

**Note:** For a four terminal MOSfet in which the bulk terminal is connected to the global net, use the formatting function corresponding to NMOSfet or PMOSfet and not MOSfet.
How CDL Out Translates Global Signals

CDL Out interprets a signal that ends with a ! as a global signal. The signal name appears in the .global and .pin statements. If the global signal is associated with a component, the signal does not appear in the I/O signal list of a subcircuit definition.

Do not use global signals with a hierarchical input connector unless the schematic is the top-level block of the design. If you use the global signal with a hierarchical input connector, the signal appears in the I/O signal list of a subcircuit definition and creates warnings when you run LOGLVS. Assign wire or label connections directly to instances in the schematic (for example, when you use globals with components.)

For nmos, pmos, and cap devices in the Cadence Library, by default a substrate connection to power or ground is made in the netlist. To override the VDD and GND defaults for these connections, include the following declarations in the .simrc file:

```
hnlCDLNMOSSBulkNetName="cvss!"
hnlCDLPMOSBulkNetName="cvdd!"
hnlCDLCAPBulkNetName="dvss!"
```

In this example, cvss is the bulk connection for the nmos device, cvdd is the bulk connection for the pmos device, and dvss is the bulk connection for the cap device. The signals appear as CVSS!, CVDD!, and DVSS! in the netlist.

Just like in the schematic where the global signal names are suffixed with !, the global signal names in the output CDL netlist file also end with !.
CDL Out Formats

The CDL view for primitive cells must contain hnlCDLFormatInst, hnlCDLParamList and hnlCDLElementSubType in its property list. This section documents the CDL Out formats for the primitive components generated by CDL Out and the cdslib components that use them.

BJT Element Format

Used by npns and pnps component types.

C InstanceName C B E [SUB] hnlCDLElementSubType $EA=0 area $L=@l $W=@w ($SUB=@sub) @off @ic @dm @m

BSIM3SOI Element Format

Used by BSIM3SOI component types.

M<name> <G> <S> <E> <model> l=@l w=@w ad=@ad as=@as pd=@p ps=@ps nrs=@nrs nrd=@nrd nrb=@nrb @off bjtoff=@bjtoff ic=@ic rtho=0 rtho ctho=0 debug=0 debug nbc=@nbc naeg=@naeg pdbcp=@pdbc psbcp=@psbc agbcp=@agbc aebcp=@aebc vbsusr=@vbsusr tnodeout=@tnodeout

Cap Element Format

Used by cap component types.

C InstanceName Y global_gnd @c m=@m [$hnlCDLElementSubType]($SUB=@sub) @ns @tc1 @tc2 @scale @cjc @ic

Capacitor Element Format

Used by capacitor and pcapacitor component types.

C InstanceName PLUS MINUS @c m=@m [$hnlCDLElementSubType]($SUB=@sub) @ns @tc1 @tc2 @scale @cj ic @ic

Diode Element Format

Used by diode and pdiode component types.

D InstanceName PLUS MINUS mname AREA=0 area PJ=pj{m=multiplier}($SUB=@sub) w=@w l=@l wp=@wp lp=@lp wm=@wm @off @ic

Inductor Element Format

Used by inductor component types.
LInstanceName PLUS MINUS @l @tc1 @tc2 @nt ic=@ic

JFET Element Format

Used by njfet and pjfet component types.

JInstanceName D G S cellName w=@w l=@l off ic=@ic m=@m

MOSFET Element Format

Used by sdalib ndepl, nfet, nsftn, pdepl, pfet, and psftn component types.

MInstanceName D G S B hnlCDLElementSubType w=@w l=@l ad=@ad as=@as pd=@pd ps=@ps nrd=@nrd nrs=@nrs @off ic=@ic m=@m $LDD[@LDD] ($NONSWAP)

NMOSFET Element Format

Used by nxfr, Nmos, nmosd, and nmos component types.

MInstanceName D G S global_gnd, hnlCDLElementSubType w=@w l=@l ad=@ad as=@as pd=@pd ps=@ps nrd=@nrd nrs=@nrs @off ic=@ic m=@m $LDD[@LDD]

NPN Element Format

Used by npn component types.

QInstanceName C B E hnlCDLElementSubType M=@m $EA=@area

PMOSFET Element Format

Used by pxfr, Pmos, pmosd, and pmose component types.

MInstanceName D G S global_Vdd hnlCDLElementSubType w=@w l=@l ad=@ad as=@as pd=@pd ps=@ps nrd=@nrd nrs=@nrs @off ic=@ic m=@m $LDD[@LDD]

PNP Element Format

Used by pnp component types.

QInstanceName C B E hnlCDLElementSubType M=@m $EA=@area

Res Element Format

Used by res component types.

RInstanceName A Y @ns | @r $[hnlCDLElementSubType] $SUB=@sub $w=@w $l=@l @ns @tc1 @tc2 @scale @rsh ac=@ac m=@m
Resistor Element Format

Used by resistor component types.

RInstanceName PLUS MINUS @ns | r [hn1CDLElementSubType] m=@m ($SUB=@sub) $w=@w
$l=@l @ns @tc1 @tc2 @scale @rsh ac=@ac ($SUB=@sub)

Transmission Line Element Format

Used by tline component types.

TInstanceName N1 N2 N3 N4 z0=@z0 td=@td f=@f nl=@nl ic=@ic

Voltage Source Element Format

Used by Vsrc element.

V<InstanceName> N+ N- @DCValue @TRAN Value @ACMag @ACPhase

Voltage Controlled Current Source Element Format

Used by VCIsrc element.

G<InstanceName> N+ N- NC+ NC- @value

Voltage Controlled Voltage Source Element Format

Used by VCVsrc element.

E<InstanceName> N+ N- NC+ NC- @value

Current Controlled Current Source Element Format

Used by ICIsrc element.

F<InstanceName> N+ N- V<controlNum> @value

Specify the controlNum property at the instance or the cdl view of the master. The instance property takes higher precedence.

Current Controlled Voltage Source Element Format

Used by ICVsrc element.

H<InstanceName> N+ N- V<controlNum> @value
Specify the `controlNum` property at the instance or the `cdl` view of the master. The instance property takes higher precedence.

**Current Source Element Format**

Used by `Isrc` element.

```
I<InstanceName> N+ N- @DCValue @TRANValue @ACMag @ACPhase
```

**CDL Out Output Files**

Examples earlier in this chapter show what the output Design Framework II file looks like after translation. Depending on what output files you specified in the *Virtuoso® CDL Out* form or template file, CDL Out produces one or more of the following files. CDL Out writes all messages to the `si.log` file.

- Error messages, preceded by “*Error*”, indicate serious, unrecoverable conditions. The converted file is incorrect. For example, if a cell instance is called but its master cell does not exist, CDL Out writes an error message to the `si.log` file.

- Warning messages preceded by “*Warning*” indicate unexpected but recoverable conditions. The converted file is usable. For example, if a string is too long and has been truncated, CDL Out writes a warning message to the `si.log` file.

- Information messages, preceded by “*Info*”, indicate the status of a process that is running or the results of a completed process. Information messages are written to the file `si.log`.

- Statistical messages that CDL Out generates as it runs are also written to the `si.log` file. You can use these messages to estimate the size of the file based on information such as the number of cells and terminals.
How to Customize CDL Netlist

CDL Out defines various hnl functions and variables, which control the CDL Out netlist format. If these functions and variables are already defined, CDL Out does not overwrite them. You can override the value of variables or functions used in CDL Out by defining them before invoking CDL Out. This can be done by loading a SKILL file before running CDL Out or by defining the variables in .simrc or .cdsinit file. To see details about the various variables and functions that can be overridden, refer to Chapter 5, Open Simulation System Reference.

Example

The variable hnlCommentStr defines the string that is placed at the beginning of a comment in the output netlist file. By default, ‘*’ is used by CDL Out to represent the comments. If you put the value of hnlCommentStr variable to ‘>’ in the .simrc file, then the comments will start with ‘>’.
CDL OUT SKILL Functions

**hnICDLPrintBJTElement**

hnICDLPrintBJTElement()

**Description**

Prints the CDL syntax of an instance of the BJT element in the netlist in the following format:

QInstanceName C B E [SUB] cellName $EA=@area $L=@l $W=@w {$SUB=@sub}
@off ic=@ic m=@m

If you create a library element similar to the BJT element, set the following property in the CDL view of the element:

hnICDLFormatInst = hnICDLPrintBJTElement()

**Arguments**

None

**Value Returned**

None
**hnICDLPrintGeneralElement**

hnICDLPrintGeneralElement()

**Description**

Prints the CDL syntax of an instance of any general element. It prints the CDL syntax in the following format:

*InstanceName O1 O2 ... I1 I2 ... OT1 OT2 ...*

where,

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O1 O2...</td>
<td>Nets on output terminals</td>
</tr>
<tr>
<td>I1 I2...</td>
<td>Nets on input terminals</td>
</tr>
<tr>
<td>OT1 OT2...</td>
<td>Nets on other terminals</td>
</tr>
</tbody>
</table>

To use this function, set the following property in the CDL view of the element:

hnICDLFormatInst = hnICDLPrintGeneralElement()

**Arguments**

None

**Value Returned**

None
**hnICDLPrintICIsrcElement**

hnICDLPrintICIsrcElement()

**Description**

Prints the CDL syntax of an instance of an ICIsrc element in the netlist in the following format:

```
FInstanceName N+ N- VcontrolVoltage @value
```

If you create a library element similar to the ICIsrc element, set the following property in the CDL view of the element:

```
hnICDLFormatInst = hnICDLPrintICIsrcElement()
```

**Arguments**

None

**Value Returned**

None
**hnICDLPrintICVsrcElement**

hnICDLPrintICVsrcElement()

**Description**

Prints the CDL syntax of an instance of an ICVsrc element in the netlist in the following format:

\[ \text{HInstanceName} \ N+ \ N- \ V\text{controlVoltage} @\text{value} \]

If you create a library element similar to the ICVsrc element, set the following property in the CDL view of the element:

hnICDLMFormatInst = hnICDLPrintICVsrcElement()

**Arguments**

None

**Value Returned**

None
**hnICDLPrintCds_Thru**

hnICDLPrintCds_Thru()

**Description**

Prints the CDL syntax of an instance of a Cds_Thru element in the netlist in the following format:

\[
\text{RInstanceName src dst @ns/100.0m $[cellName]}
\]

If you create a library element similar to the Cds_Thru element, set the following property in the CDL view of the element:

hnICDLFormatInst = hnICDLPrintCds_Thru()

**Arguments**

None

**Value Returned**

None
**hnICDLPrintInductorElement**

hnICDLPrintInductorElement()

**Description**

Prints the CDL syntax of an instance of an Inductor element in the netlist in the following format:

```
LInstanceName PLUS MINUS @l @tc1 @tc2 @nt ic=@ic
```

If you create a library element similar to the Inductor element, set the following property in the CDL view of the element:

```
hnICDLFormatInst = hnICDLPrintInductorElement()
```

**Arguments**

None

**Value Returned**

None
**hnICDLPrintIsrcElement**

hnICDLPrintIsrcElement()

**Description**

Prints the CDL syntax of an instance of an Isrc element in the netlist in the following format:

```
InstanceName N+ N- @DCValue @TRANValue @ACMag @ACPhase
```

If you create a library element similar to the Isrc element, set the following property in the CDL view of the element:

```
hnICDLFormatInst = hnICDLPrintIsrcElement()
```

**Arguments**

None

**Value Returned**

None
**hnlCDLPrintJfetElement**

hnlCDLPrintJfetElement()

**Description**

Prints the CDL syntax of an instance of a *Jfet* element in the netlist in the following format:

```
JInstanceName D G S cellName w=@w l=@l @off ic=@ic m=@m
```

If you create a library element similar to the *Jfet* element, set the following property in the CDL view of the element:

```
hnlCDLFormatInst = hnICDLPrintJfetElement()
```

**Arguments**

None

**Value Returned**

None
hnICDLPrintNMOSfetElement

hnICDLPrintNMOSfetElement()

Description

Prints the CDL syntax of an instance of a NMOSfet element in the netlist in the following format:

MInstanceName D G S global gnd, cellName w=@w l=@l ad=@ad as=@as pd=@pd ps=@ps nrd=@nrd nrs=@nrs @off ic=@ic m=@m $LDD[@LDD]

If you create a library element similar to the NMOSfet element, set the following property in the CDL view of the element:

hnICDLFormatInst = hnICDLPrintNMOSfetElement()

Arguments

None

Value Returned

None
hnICDLPrintNPNElement

hnICDLPrintNPNElement()

**Description**

Prints the CDL syntax of an instance of a NPN element in the netlist in the following format:

\[ Q\text{InstanceName} \ C \ B \ E \ cellName \ M=@m \ $EA=@area \]

If you create the library element similar to the NPN element, set the following property in the CDL view of the element:

hnICDLFormatInst = hnICDLPrintNPNElement()

**Arguments**

None

**Value Returned**

None
**hnICDLPrintPMOSfetElement**

hnICDLPrintPMOSfetElement()

**Description**

Prints the CDL syntax of an instance of a PMOSfet element in the netlist in the following format:

```
M InstanceName D G S global_Vdd cellName w=@w l=@l ad=@ad as=@as pd=@pd ps=@ps nrd=@nrd nrs=@nrs @off ic=@ic m=@m $LDD[@LDD]
```

If you create a library element similar to the PMOSfet element, set the following property in the CDL view of the element:

```
hnICDLFormatInst = hnICDLPrintPMOSfetElement()
```

**Arguments**

None

**Value Returned**

None
**hn1CDLPrintPNPElement**

*hn1CDLPrintPNPElement()*

**Description**

Prints the CDL syntax of an instance of a PNP element in the netlist in the following format:

\[ Q \text{InstanceName} C B E \text{cellName} M=@m \$EA=@area \]

If you create a library element similar to the PNP element, set the following property in the CDL view of the element:

*hn1CDLFormatInst = hn1CDLPrintPNPElement()*

**Arguments**

None

**Value Returned**

None
hnICDLPrintResistorElement

hnICDLPrintResistorElement()

Description

Prints the CDL syntax of an instance of a Resistor element in the netlist in the following format:

\[
R \text{InstanceName PLUS MINUS @ns} | @r [cellName] m=@m \{\text{SUB}@sub\} \ w=@w \ l=@l \ @ns \ @tc1 \ @tc2 \ @scale \ @rsh \ \text{ac}=@ac \ \{\text{SUB}@sub\}
\]

If you create a library element similar to the Resistor element, set the following property in the CDL view of the element:

hnICDLFormatInst = hnICDLPrintResistorElement()

Arguments

None

Value Returned

None
**hn1CDLPrintSchottkyTranElement**

hn1CDLPrintSchottkyTranElement()

**Description**

Prints the CDL syntax of an instance of a SchottkyTran element in the netlist in the following format:

\[
C \text{InstanceName PLUS MINUS @c \{cellName\} @ns @tc1 @tc2 @scale @cj ic=@ic m=@m \{\$SUB=@sub\} Q name.1 C B E @NP cllname Q name.2 B C cellname
\]

If you create a library element similar to the SchottkyTran element, set the following property in the CDL view of the element:

hn1CDLFormatInst = hn1CDLPrintSchottkyTranElement()

**Arguments**

None

**Value Returned**

None
**hnlCDLPrintTlineElement**

hnlCDLPrintTlineElement()

**Description**

Prints the CDL syntax of an instance of a Tline element in the netlist in the following format:

$$T_{\text{InstanceName}} N1 N2 N3 N4 z0=@z0 \text{ td}=@td \text{ f}=@f \text{ nl}=@nl \text{ ic}=@ic$$

If you create a library element similar to the Tline element, set the following property in the CDL view of the element:

hnlCDLFormatInst = hnlCDLPrintTlineElement()

**Arguments**

None

**Value Returned**

None
**hnCDLPrintVCIsrcElement**

hnCDLPrintVCIsrcElement()

**Description**

Prints the CDL syntax of an instance of a **VCIsrc** element in the netlist in the following format:

\[ \text{GInstanceName} \, N+ \, N- \, NC+ \, NC- \, @value \]

If you create a library element similar to the **VCIsrc** element, set the following property in the CDL view of the element:

\[ \text{hnCDLFormatInst} = \text{hnCDLPrintVCIsrcElement()} \]

**Arguments**

None

**Value Returned**

None
**hnICDLPrintVCVsrcElement**

*hnICDLPrintVCVsrcElement()*

**Description**

Prints the CDL syntax of an instance of a VCVsrc element in the netlist in the following format:

\[ E \text{InstanceName} \ N+ \ N- \ NC+ \ NC- \ @value \]

If you create a library element similar to the VCVsrc element, set the following property in the CDL view of the element:

*hnICDLFormatInst* = *hnICDLPrintVCVsrcElement()*

**Arguments**

None

**Value Returned**

None
**hnlCDLPrintVsrcElement**

*hnlCDLPrintVsrcElement()*

**Description**

Prints the CDL syntax of an instance of a *Vsrc* element in the netlist in the following format:

\[ VInstanceName \ N+ \ N- \ @DCValue \ @TRANValue \ @ACMag \ @ACPhase \]

If you create a library element similar to the *Vsrc* element, set the following property in the CDL view of the element:

*hnlCDLFormatInst = hnlCDLPrintVsrcElement()*

**Arguments**

None

**Value Returned**

None
hnlCDLPrintMultiCNPNElement

hnlCDLPrintMultiCNPNElement()

Description

Prints the CDL syntax of an instance of a MultiCNPN element in the netlist in the following format:
QInstanceName.1 C1 B E cellName
  QInstanceName.n Cn B E cellName

If you create a library element similar to the MultiCNPN element, set the following property in the CDL view of the element:
hnlCDLFormatInst = hnlCDLPrintMultiCNPNElement()

Arguments

None

Value Returned

None
**hn1CDLPrintMultiCPNPElement**

`hn1CDLPrintMultiCPNPElement()`

**Description**

Prints the CDL syntax of an instance of a `MultiCPNP` element in the netlist in the following format:

```
Q_InstanceName.1 C1 B E cellName
Q_InstanceName.n Cn B E cellName
```

If you create a library element similar to the `MultiCPNP` element, set the following property in the CDL view of the element:

```
hn1CDLFormatInst = hn1CDLPrintMultiCPNPElement()
```

**Arguments**

None

**Value Returned**

None
**hnICDLPrintMultiENPNEElement**

hnICDLPrintMultiENPNElement()

**Description**

Prints the CDL syntax of an instance of a MultiENPN element in the netlist in the following format:

QInstanceName.1 C1 B E cellName

QInstanceName.n Cn B En cellName

If you create a library element similar to the MultiENPN element, set the following property in the CDL view of the element:

hnICDLFormatInst = hnICDLPrintMultiENPNElement()

**Arguments**

None

**Value Returned**

None
hn1CDLPrintMultiEPNPElement

hn1CDLPrintMultiEPNPElement()

Description

Prints the CDL syntax of an instance of a MultiEPNP element in the netlist in the following format:

QInstanceName.1 C1 B E cellName
.
QInstanceName.n Cn B En cellName

If you create a library element similar to the MultiEPNP element, set the following property in the CDL view of the element:

hn1CDLFormatInst = hn1CDLPrintMultiEPNPElement()

Arguments

None

Value Returned

None
**hnlCDLPrintCapElement**

hnlCDLPrintCapElement()

**Description**

Prints the CDL syntax of an instance of a cap element in the netlist in the following format:

```
CInstanceName Y global_gnd @m=@m $[cellName] {$SUB=@sub} @ns @tcl
@tc2 @scale @cjc=ic @area=area l=l w=w
```

If you create a library element similar to the cap element, set the following property in the CDL view of the element:

```
hnlCDLFormatInst = hnlCDLPrintCapElement()
```

**Arguments**

None

**Value Returned**

None
hn1CDLPrintCapacitorElement

hn1CDLPrintCapacitorElement()

Description

Prints the CDL syntax of an instance of a capacitor element in the netlist in the following format:

\[ C\{InstanceName PLUS MINUS @cm=@m $[cellName] \{SUB=@sub\}@ns @tc1 @tc2 @scale @cjic=@ic area=@area l=@l w=@w \] \]

If you create a library element similar to the capacitor element, set the following property in the CDL view of the element:

hn1CDLFormatInst = hn1CDLPrintCapacitorElement()

Arguments

None

Value Returned

None
**hn1CDLPrintDiodeElement**

hn1CDLPrintDiodeElement()

**Description**

Prints the CDL syntax of an instance of a diode element in the netlist in the following format:

```
DiInstanceName PLUS MINUS mname AREA=@area PJ=@pj \{m=multiplier\} \{$SUB=@sub\} w=@w l=@l wp=@wp lp=@lp wm=@wm @offic=@ic
```

If you create a library element similar to the diode element, set the following property in the CDL view of the element:  
```
hn1CDLFormatInst = hn1CDLPrintDiodeElement()
```

**Arguments**

None

**Value Returned**

None
hn1CDLPrintBSIM3SOIElement

hn1CDLPrintBSIM3SOIElement()

Description

Prints the CDL syntax of an instance of a BSIM3SOI element in the netlist in the following format:

M<name> <D> <G> <S> <E> <model> l=@l w=@w ad=@ad as=@as pd=@pd ps=@ps nrs=@nrs nrd=@nrd nrb=@nrb @off bjtoff=@bjtoff ic=@ic rtho=@rtho ctho=@ctho debug=@debug nbc=@nbc nseg=@nseg pdbcp=@pdbcp psbcp=@psbcp agbcp=@agbcp aebcp=@aebcp vbsusr=@vbsusr tnodeout=@tnodeout

If you create a library element similar to the BSIM3SOI element, set the following property in the CDL view of the element:

hn1CDLFormatInst = hn1CDLPrintBSIM3SOIElement()

Arguments

None

Value Returned

None
hnlCDLPrintResElement

hnlCDLPrintResElement()

Description

Prints the CDL syntax of an instance of a Res element in the netlist in the following format:
RInstanceName A Y @ns | @r [$cellName] $SUB=@sub $w=@w $l=@l @ns @tc1 @tc2 @scale @rsh @c= @ac m=@m

If you create a library element similar to the Res element, set the following property in the CDL view of the element:
hnlCDLFormatInst = hnlCDLPrintResElement()

Arguments

None

Value Returned

None
transCdlOutDisplay

transCdlOutDisplay(
    transCdlOutForm
)

Description

Invokes the CDL Out GUI form.

Arguments

transCdlOutForm Specifies to invoke the CDL Out form.

Value Returned

None
Using SKILL to Customize Stream and CIF Files

This chapter describes the following:

- Overview on page 270
- Customizing the Graphical User Interface of Translators on page 270
- Customizing Input Files on page 271
- Customizing Export Files on page 279
Overview

The SKILL functions described in this chapter are intended to be defined as SKILL procedures. The arguments for the SKILL functions are actually variables for the data that the translator supplies as it runs. The data type defined for each argument applies to the data that the translator supplies for the argument, not to the form of the input you type in the SKILL file.

For more information about writing SKILL routines, refer to the SKILL Language User Guide and SKILL Language Reference.

Customizing the Graphical User Interface of Translators

The SKILL function described in this section helps you control the user interface forms of translators at will. You might need it when you want to set key bindings for your own use.

pipodisplay

pipodisplay( s_formHandle )

=> t

Description

Brings up a window containing the translator user interface form specified by s_formHandle.

Arguments

s_formHandle  The translator form name, which can be one of the following:
transStreamInForm, transCifInForm,
transStreamOutForm, transCifOutForm

Value Returned

t  Returns t when one of the pipo translator user interface forms is successfully returned.
Example

The following example brings up a window containing the Virtuoso® Stream Out form.

```
pipoDisplay( transStreamOutForm )
```

Customizing Input Files

You can use the SKILL procedures described in this section to map cell names, layer number, and text fonts from Stream, and Caltech Intermediate Format (CIF) input files to the Design Framework II database. You can also use SKILL procedures to control how the translators translate illegal shapes.

Before you use a SKILL file to customize translation, note the following:

- If you use mapping tables to control how the translator translates cell names, layers, and text fonts, the translator might not run the SKILL file. For information about when the translator uses the SKILL file, refer to the map table field descriptions in “Using Map Files When Importing” on page 26.
- You can use only one SKILL file with a translator, so include all the SKILL procedures you want to use in a single file.
- The procedures listed in this section are started only when you use a translator.
- All procedures must follow the syntax shown in this section. If a procedure cannot be loaded, a warning message appears. Translation stops or is incorrect.
- System performance slows when you use these SKILL procedures.

```
piCellNameMap
piCellNameMap( t_cell )
=> l_cellView / nil
```

Description

Passes cell and view names for each cell name in the input file to a user-defined procedure.

*Import – Stream, Import – CIF* interprets the output of the procedure in the same way as it interprets the Design Framework II cell and view names.
Arguments

t_cell

The input cell name.

Value Returned

l_cellView

A list containing the Design Framework II cell and view names.

nil

No cell and view names are found.

Example

In this example, the characters "_pi" are added to every cell name.

```
procedure( piCellNameMap( cell )
    prog( ( mapname )
        sprintf( mapname "%s_pi" cell )
        return( list( mapname "layout" ))
    );prog
);procedure
```

piLayerMap

```
piLayerMap( x_layer x_datatype ) => l_lpp / nil
```

Description

Passes the layer number and datatype for each layer in the input Stream file to a user-defined procedure.

*Import – Stream* interprets the output of the procedure in the same way as it interprets the Design Framework II layer name and purpose.

This SKILL function is not used by *Import – CIF*.

Arguments

x_layer

The input layer number.

x_datatype

The input data type number.
Value Returned

$l_1pp$

A list containing two strings. The first string is the Design Framework II layer name. The second string is the Design Framework II layer-purpose.

nil

No layer or data type numbers are found.

Example

In this example, the input layers 1 to 10 are mapped to the Design Framework II layer-purpose pair text drawing.

```skill
procedure( piLayerMap( layer datatype )
    prog( ( lay )
        if( lay>=1 && lay <=10 then
            return( list( "text" "drawing" )
                if( lay == 16 return( list( "diff" "drawing" )
                if( lay == 30 return( list( "substrate" "drawing" )
            return( list( "nwell" "drawing" )
        );prognilnilnil
    );procedure
```

piCifLayerMap

```skill
piCifLayerMap( t_layerName )
    => l_1pp / nil
```

Description

Passes the CIF format layer name for each layer in the input file to a user-defined procedure.

Import – CIF interprets the output of the procedure in the same way as it interprets the Design Framework II layer name and purpose.

Arguments

`t_layerName` The layer name to map.

Value Returned

$l_1pp$

A list containing two strings. The first string is the Design Framework II layer name. The second string is the Design Framework II layer-purpose.
nil

No layer is found.

Example

In this example, the CIF layer names are mapped to the Design Framework II layer name. For example, layer NM is mapped to layer ntub. Each of these layers is assigned drawing as its purpose.

```prolog
procedure( piCifLayerMap( layer_name )
  prog( ( lay )
    case( layer_name
      ("NM" lay="ntub")
      ("NP" lay="thinox")
      ("MT" lay="diff")
      (t lay="substrate"))
    return( list( lay "drawing" ) )
  );prog
);procedure
```
piTextMap

piTextMap(
    t_label
)
=> t_changedLabel

Description

Modifies the text that is translated from the Stream file to a dfll library.

During Stream In, if you specify a SKILL file having piTextMap() defined in it, then this
function is called for each text object and the text string is passed as an argument to this
function. The string returned by this function is used to modify the text in the target dfll library.

Arguments

$t_label$ Text to be modified

Value Returned

$t_changedLabel$ Modified text

Example

During Stream In, you can replace the character [ with < and character ] with > by using the
piTextmap() function.

procedure( piTextMap( label)
    prog( ( newLabel )
        rexCompile( "\\["]
        newLabel = rexReplace( label "<" 1)
        rexCompile( "\\]" )
        newLabel = rexReplace( newLabel ">
        return( newLabel )
    ); prog
}
textFontMap

textFontMap( x_font )
    => l_fontType / nil

Description

Passes each font number in the input file to a user-defined procedure.

*Import – Stream, Import – CIF* interprets the output of the procedure in the same way as it interprets the Design Framework II font number and height.

Arguments

x_font

The input font number.

Value Returned

l_fontType

A list containing the Design Framework II font type and optionally the font height. If the system outputs a single number, it is the Design Framework II font type and indicates that the font height did not change.

nil

No font number found.

Example

In this example, the Stream fonts 0, 1, and 2 map to dfII fonts 3, 4, and 6. These fonts use the original font height.

```skill
procedure( textFontMap( fontNum )
    prog( ()
        case( fontNum
            (0 return( 3 ) )
            (1 return( 4 ) )
            (3 return( 6 ) )
            (t return( fontNum ))
        );case
    );prog
);procedure
```
Note: The internal representation of the dfII fonts are:

<table>
<thead>
<tr>
<th>dfII Font Name</th>
<th>dfII Font Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbcEuroStyleFont</td>
<td>1</td>
</tr>
<tr>
<td>dbcGothicFont</td>
<td>2</td>
</tr>
<tr>
<td>dbcMathFont</td>
<td>3</td>
</tr>
<tr>
<td>dbcRomanFont</td>
<td>4</td>
</tr>
<tr>
<td>dbcScriptFont</td>
<td>5</td>
</tr>
<tr>
<td>dbcStickFont</td>
<td>6</td>
</tr>
<tr>
<td>dbcFixedFont</td>
<td>7</td>
</tr>
<tr>
<td>dbcSwedishFont</td>
<td>8</td>
</tr>
</tbody>
</table>

**pipoErrShapesHandler**

```clojure
pipoErrShapesHandler( x_errcode t_shape )
    => x_transcode / nil
```

Description

Passes the error condition and type of shape for each illegal shape in the import file to a user-defined procedure.

*Import – Stream, Import – CIF* interprets the output of the procedure in the same way as it interprets an integer that defines how the translator translates an illegal shape.

Arguments

- **x_errcode**
  - The error condition of the illegal shape.
  - Valid Value: 0 (for zero-area shapes)

- **t_shape**
  - The type of illegal shape.
  - Valid Values: "rectangle" "polygon"

Global Variables

Two global variables are available for your user-defined procedure.
**l_pipoArray**

A list containing all the points of the shape. The translator translates the points of the illegal shape to a list and sets the list as the value of this variable for the SKILL function. If you choose to translate the shape, the translator uses this variable to define the shape to translate. You can reset the value of this variable, if you want. The following is an example of the points list:

```plaintext
l_pipoArray = list( 1:1 4:4 ) ; rectangle
l_pipoArray = list( 1:1 2:2 2:3 1:3 1:1 ) ; polygon
```

All the coordinates of the points are in user units.

**t_pipoMsg**

A string containing the user-defined message that you want to print out in the error file. If the return code is 0 or 254, the translator prints this string in the error file as a warning message. For example, if the message is

```plaintext
t_pipoMsg="Error rectangle is corrected.\n"
```

the printed message in the error file is

```
* Warning * Error rectangle is corrected.
```

**Value Returned**

**x_transcode**

An integer that indicates how the translator translates the illegal shape.

- **0** Translates the shape defined by the `l_pipoArray` global variable, and prints the contents of the `t_pipoMsg` global variable in the error file.
- **254** Does not translate the shape, but prints the contents of `t_pipoMsg` in the error file.
- **255** Does not translate the shape, and ignores the `t_pipoMsg` global variable.

**nil**

No errors or invalid shapes found.

**Example**

This example tells the translator to

- Correct a zero-area rectangle and give a new points array and a user-defined message
- Drop rectangles that contain errors other than zero-area and issue a user-defined message
- Drop all shapes that contain errors

```scheme
procedure( pipoErrShapesHandler( errcode shape )
  prog( ( l1 l2 )
    case( shape
      ("rectangle" ;if the error shape is rectangle
        if( errcode==0 then ;and it’s a zero-area rectangle
          pipoArray = userCorrectRect() ;after user corrected it
            ;this array will be taken as the
            ;new BBOX of the rectangle
          l1 = car( pipoArray )
          l2 = cadr( pipoArray )
          sprintf( pipoMsg "zero-area rectangle was corrected,
            ;with \n            new BBOX = (%g %g)(%g %g).
          " car(l1) cadr(l1) car(l2) cadr(l2))
          return( 0 )
        );end of if
        sprintf( pipoMsg "Do not handle rectangle with \n          error code %d.\n        " errcode )
        return( 254 )
      );rectangle
    (t ;this will drop all the other shapes
      return( 255 ) ) ;this will drop all the other shapes
    );end of case
  ))
```

## Customizing Export Files

You can use the SKILL routines described in this section to map cell names, parameterized cell (pcell) names, layer numbers, and text fonts from the Design Framework II design to the export file. You can also use SKILL routines to control how the Stream and CIF export translators translate illegal shapes.

Before you use a SKILL file to customize exportation, note the following:

- If you use mapping tables for cell names, layers, and text fonts, *Export – Stream* and *Export – CIF* might not read the SKILL file. For more information about when the translators use SKILL procedures, refer to “Using Map Files when Exporting” on page 36.
- You can use only one SKILL file, so include all the SKILL procedures you want to use in a single file.
- The procedures listed in this section are started only when you run the export translator.
All procedures must follow the syntax shown in this section. If a procedure cannot be loaded, a warning message appears and translation stops or is incorrect.

System performance slows when you use these SKILL procedures.

**poCellNameMap**

```skill
poCellNameMap( t_lib t_cell t_view )
  => t_mapName / nil
```

**Description**

Passes the cell, view, and library name for each cell in the Design Framework II library to a user-defined procedure.

*Import – Stream* and *Import – CIF* interpret the output of the procedure in the same way as it interprets the translated cell name.

**Arguments**

- **t_lib**
  - The Design Framework II library name.

- **t_cell**
  - The Design Framework II cell name.

- **t_view**
  - The Design Framework II view name.

**Value Returned**

- **t_mapName**
  - The translated cell name.

- **nil**
  - No cells found.

**Example**

In this example, the characters "_po" are added to every cell name. The library and view are ignored.

```skill
procedure( poCellNameMap( lib cell view )
  prog( ( mapname )
    sprintf( mapName "%s_po" cell)
    return( mapName )
  );prog
);procedure
```
**poCifLayerMap**

```skill
poCifLayerMap( t_layerName t_layerPurpose ) => t_layerName
```

**Description**

Passes a Design Framework II format layer-purpose pair for each layer in the input file to a user-defined procedure.

*Import – CIF* interprets the output of the procedure in the same way as it interprets the CIF layer name.

**Arguments**

- `t_layerName` The Design Framework II layer name.
- `t_layerPurpose` The Design Framework II layer-purpose.

**Value Returned**

- `t_layerName` The CIF layer name.

**Example**

In this example, each Design Framework II layer name is mapped to an output layer name. The case `t` maps all other layers that are not listed in the `case` statement to the CIF layer "OTHER".

```skill
procedure( poCifLayerMap( layerName purpose )
    prog( ( cifName )
        case( layerName
            ("thinox" return( "THIN" )
            ("diff" return( "MET1" )
            ("metal" return( "MET2" )
            (t return( "OTHER" )
        );case
    );prog
);procedure
```

**textFontMap**

```skill
textFontMap( x_font ) => l_mapFont / nil
```
Description

Passes each font type in the Design Framework II library to a user-defined procedure.

**Export – Stream and Import – CIF** interpret the output of the procedure in the same way as it interprets the translated font type.

Arguments

$x_{font}$ The Design Framework II font type.

Value Returned

$l_{mapFont}$ A list containing one or two integers. The first integer is the translated font type. The second integer is the translated font height. If the return value is a single integer, it is the font type and indicates that the font height did not change.

nil No font type found.

Example

In this example, the dfII fonts 4, 5, and 6 are mapped to Stream fonts 0, 1, and 2. All others are mapped to Stream font 0.

```skill
procedure( textFontMap( font )
    prog( ()
        case( font
            (4   return(0))
            (5   return(1))
            (6   return(2))
            (t   return(0))
        );case
    );prog
);procedure
```

**Note:** Default font mapping for PIPO Stream Out is:

<table>
<thead>
<tr>
<th>dfII Font Name</th>
<th>Stream Font Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbcStickFont</td>
<td>0</td>
</tr>
<tr>
<td>dbcEuroStyleFont</td>
<td>1</td>
</tr>
</tbody>
</table>
The internal representation of dfII fonts are:

<table>
<thead>
<tr>
<th>dfII Font Name</th>
<th>dfII Font Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbcEuroStyleFont</td>
<td>1</td>
</tr>
<tr>
<td>dbcGothicFont</td>
<td>2</td>
</tr>
<tr>
<td>dbcMathFont</td>
<td>3</td>
</tr>
<tr>
<td>dbcRomanFont</td>
<td>4</td>
</tr>
<tr>
<td>dbcScriptFont</td>
<td>5</td>
</tr>
<tr>
<td>dbcStickFont</td>
<td>6</td>
</tr>
<tr>
<td>dbcFixedFont</td>
<td>7</td>
</tr>
<tr>
<td>dbcSwedishFont</td>
<td>8</td>
</tr>
</tbody>
</table>

**Description**

Passes the error condition and type of shape for each illegal shape in the import file to a user-defined procedure.

*Export – Stream, Import – CIF* interpret the output of the procedure in the same way as it interprets an integer, which defines how the translator translates the illegal shape.

**Arguments**

- **x_errcode**
  - The error condition of the illegal shape.
  - Valid Value: 0 (for zero-area shapes)
Design Data Translator's Reference
Using SKILL to Customize Stream and CIF Files

\[ t\_shape \]

The type of illegal shape.
Valid Values: "rectangle" "polygon"

Global Variables

\[ l\_pipoArray \]
A list containing all the points of the shape. The translators translate the points of the illegal shape to a list and set the list as the value of this variable for the SKILL function. If you choose to translate the shape, the translator uses this variable to define the shape to translate. You can reset the value of this variable, if you want.

The following is an example of the points list:

\[
\begin{align*}
\text{pipoArray} &= \text{list( 1:1 4:4 ) (rectangle)} \\
\text{pipoArray} &= \text{list( 1:1 2:2 2:3 1:3 1:1 ) (polygon)}
\end{align*}
\]

Note: In the SKILL function, all the coordinates of the points are in user units.

\[ t\_pipoMsg \]
An optional string containing the message that you want to print out in the error file. If the return code is 0 or 254, the translator prints it in the error file as a warning message. For example, if the message is

\[
pipoMSG="\text{Error rectangle is corrected.}\n"
\]

the printed message in the log file is

* Warning * Error rectangle is corrected.

Value Returned

\[ x\_transcode \]
An integer that indicates how the translator translates the illegal shape.

\[
\begin{align*}
0 & \quad \text{Translates the shape defined by the global variable } l\_pipoArray, \text{ and prints the contents of } t\_pipoMsg \text{ in the error file.} \\
254 & \quad \text{Does not translate the shape, but prints the contents of } t\_pipoMsg \text{ in the error file.} \\
255 & \quad \text{Does not translate the shape, and ignores the } t\_pipoMsg.
\end{align*}
\]

\[ \text{nil} \]
No errors or illegal shapes found.
Example

This example tells the translator to

- Correct a zero-area rectangle, create a new points array, and issue a user-defined message
- Drop rectangles that contain errors other than zero-area and issue a user-defined message
- Drop all shapes that contain errors

```
procedure(pipoErrShapesHandler(errcode shape)
  prog((l1 l2)
    case(shape
      "rectangle" ;if the error shape is rectangle
        if(errcode==0 then ;and it’s a zero-area rectangle
          pipoArray = userCorrectRect() ;after user corrected
            ;it, this array will be taken as the
            ;new BBOX of the rectangle
          l1 = car(pipoArray)
          l2 = cadr(pipoArray)
          sprintf(pipoMsg "zero-area rectangle was corrected, with \
                        new BBOX = (%g %g)(%g %g).\n" car(l1) cadr(l1) \
                        car(l2) cadr(l2))
          return(0)
        );end of if
        sprintf(pipoMsg "Do not handle rectangle with \
                    error code %d.\n" errcode)
        return(254)
      );rectangle
      (t return(255)) ;this will drop all the other shapes
        ;that passed in as if there were no SKILL
        ;function defined, which means those shapes
        ;except "rectangle" will be dropped with a
        ;pre-defined warning message
    );end of case
  );end of prog
);end of procedure
```

`poParamCellNameMap`

```
poParamCellNameMap(t_name d_cvid)
  => t_mapName / nil
```

**Description**

Passes each parameterized cell name in the output file to a user-defined procedure.

*Export – Stream* interprets the output of the procedure in the same way as it interprets the translated name for the parameterized cell.
Arguments

\[ t\_name \]
The Design Framework II parameterized cell name.

\[ d\_cvid \]
The Design Framework II parameterized cell variant cellview identifier.

Value Returned

\[ t\_mapName \]
The name of the translated parameterized cell.

\[ nil \]
No parameterized cells found.

Example

In this example, the translator appends the cellview identifier to every parameterized cell name, making the cell name unique.

```skill
procedure( poParamCellNameMap( name ID )
  prog( ( newName )
    sprintf( newName "%s_%s" name ID )
    return( newName )
  );prog
);procedure
```
**poTextMap**

```plaintext
poTextMap(
    t_label
) => t_changedLabel
```

**Description**

Modifies the text that is translated from the dfII library to the Stream file.

During Stream Out, if you specify a SKILL file having `poTextMap()` defined in it, then this function is called for all text objects and the text string is passed as an argument to this function.

The string returned by this function is used to create the text object in the Stream file. It is applicable to all the strings present in the dfII library and being translated as text in the Stream file as labels, textDisplay, and so on. The string is also applicable to the text translated corresponding to pins when you use the *Convert Pin to* option to translate pins as texts.

**Arguments**

- **t_label**: Text to be modified

**Value Returned**

- **t_changedLabel**: Modified text

**Example**

During Stream In, you can replace the character `[` with `<` and character `]` with `>` by using the `poTextmap()` function.

```plaintext
procedure( poTextMap( label)
    prog( ( newLabel )
        rexCompile( "\\[" )
        newLabel = rexReplace( label "<" 1)
        rex Compile( "\\]" )
        newLabel = rexReplace( newLabel ">" 1)
        return( newLabel )
    ); prog
)
```
Translating LEF and DEF Files

This chapter describes the following:

- Overview
- LEF and DEF Translation Commands
- LEF/DEF Translation Process
- LEF Data Map
- DEF Data Map
- SKILL Functions
Overview

You can transfer data between the dfII environment and the environment of other Place & Route (P&R) tools using Library Exchange Format (LEF) and Design Exchange Format (DEF) files. Refer to the LEF/DEF Language Reference for a description of the LEF and DEF file formats. This chapter describes the following:

- Using the CIW to translate libraries and designs to and from LEF and DEF.
- Using the command line interface to translate libraries and designs to and from LEF and DEF.
- Mapping of LEF and DEF statements to the dfII environment.

LEF and DEF Translation Commands

This section describes the LEF and DEF commands that are available in the CIW File - Import and File – Export commands. The LEF and DEF commands read and write library and design information in LEF and DEF formats.

**File – Import – DEF**

The File - Import - DEF command reads a DEF format design file into dfII. It is also referred to as DEFIn.

**File – Import – LEF**
The File - Import - LEF command reads a LEF format library description file and stores the information in dfII library. It is also referred to as LEFIn.

File – Export – DEF

The File - Export - DEF command creates a DEF format design file from a dfII design cellview. It is also referred to as DEFOut.

File – Export – LEF

The File - Export - LEF command creates a LEF format library description file from a dfII library. It is also referred to as LEFOut.

DEFINE Statements

LEF and DEF translators do not support DEFINE statements. The File – Import command resolves the DEFINE statements by replacing references to each DEFINE variable with the actual value of the variable.

ALIAS Statements

The File – Import commands try to resolve ALIAS statements.

The commands write ALIAS statements that cannot be resolved, such as SROUTE TASK statements, to macro files. Each unresolved ALIAS statement becomes a separate macro file.

An example of a macro file follows:

```
# &ALIAS &&DF08 =
save ;
set v gdsii.libread.handle.nonorthogonal.shapes ignore ;
inp gds f df08.gds o df08-phy.lef layer 7 map cut12 layer 8 map m1 layer 10 map m2 pin 19 port 8 10 pinarbitration 10 8 ;
save ;
inp lef f df08-phy.lef ;
save
# &ENDALIAS
```  

An example of an alias file follows:

```
BEGIN ALIAS_FILE
&ALIAS &&FILLCELL =
sroute addcell model fill-cell prefix fill area (7000,7000) (23000,19000)
# &ENDALIAS
```
Cell Name Collisions

The Silicon Ensemble LEF file allows the same name for macros, vias, sites, and arrays. The dfII library requires a unique name for each cell. When a LEF file gives two cells the same name, File – Import – LEF adds a dash and an object type to the LEF name to create the dfII name.

File – Export – LEF uses the dfII name and does not regenerate the original name for the LEF file.

Valid Characters

Valid characters are not identical between dfII and LEF/DEF. The following table shows how characters are mapped between LEF/DEF and dfII.

<table>
<thead>
<tr>
<th>Characters</th>
<th>LEF/DEF Original</th>
<th>dfII Mapped</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchy delimiter</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>Alphanumeric string within parentheses</td>
<td>( )</td>
<td>{ }</td>
</tr>
<tr>
<td>Numeric string within parentheses</td>
<td>()</td>
<td>()</td>
</tr>
<tr>
<td>Alphanumeric string within angle brackets</td>
<td>&lt; &gt;</td>
<td>[ ]</td>
</tr>
<tr>
<td>Numeric string within angle brackets</td>
<td>&lt; &gt;</td>
<td>&lt; &gt;</td>
</tr>
<tr>
<td>Exclamation mark as the last character</td>
<td>!</td>
<td>-</td>
</tr>
<tr>
<td>Comma</td>
<td>,</td>
<td>_</td>
</tr>
</tbody>
</table>
The following are examples of character mapping:

- Parentheses or angled brackets around numbers are not mapped.

<table>
<thead>
<tr>
<th>LEF/DEF Value</th>
<th>Mapped dfII Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>/I8/I1(13)</td>
<td></td>
</tr>
<tr>
<td>/I8/I1&lt;13&gt;</td>
<td></td>
</tr>
</tbody>
</table>

- Parentheses or angle brackets around alphanumeric strings are mapped.

<table>
<thead>
<tr>
<th>LEF/DEF Value</th>
<th>Mapped dfII Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>/I65/I1(ABC13)</td>
<td></td>
</tr>
<tr>
<td>/I65/I1&lt;ABC13&gt;</td>
<td></td>
</tr>
</tbody>
</table>

- The exclamation character is mapped only if it is the last character.

<table>
<thead>
<tr>
<th>LEF/DEF Value</th>
<th>Mapped dfII Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>/I6/I11!OP</td>
<td></td>
</tr>
<tr>
<td>/STOV/QOP!</td>
<td></td>
</tr>
</tbody>
</table>

If the dfII name of an instance is different from the original DEF name, File – Export – DEF uses the dfII name and does not regenerate the original name for the DEF file.

**Version Settings**

The versions of LEF and DEF can be set by initializing the `gec3CurrentLEFVersion` or `dfIILEFDEFCurrentVersion` variables. If both the variables are set, the `dfIILEFDEFCurrentVersion` variable takes precedence. The default value of version for LEF and DEF is 5.4. LEF/DEF reads all the constructs irrespective of the version (5.4 or 5.5) but writes only those constructs for which the version is specified.

To write LEF 5.5 and DEF 5.5 constructs, version can be set as described below:

```plaintext
dfIILEFDEFCurrentVersion="5.5"
```

or

```plaintext
gec3CurrentLEFVersion="5.5"
```
LEF/DEF Translation Process

Invoking LEF/DEF Translator

The LEF/DEF translator can be invoked from the icca, icfb, msfb, layout, and layoutPlus tools. When the tools layout, msfb, and layoutPlus are used for LEF In, LEF Out, DEF In, and DEF Out, the lefdef.exe executable runs in the background. The GUI of the LEF and DEF translators remains unchanged irrespective of the method used to invoke the LEF and DEF translators.

This topic covers the following methods of invoking the translators and also describes how to translate a compressed LEF or DEF file:

- **LEF In**
  - Running LEF In from the GUI
  - Running LEF In from the Command Line
  - Translating a Compressed LEF File

- **LEF Out**
  - Running LEF Out from the GUI
  - Running LEF Out from the Command Line

- **DEF In**
  - Running DEF In from the GUI
  - Running DEF In from the Command Line
  - Translating a Compressed DEF File

- **DEF Out**
  - Running DEF Out from the GUI
  - Running DEF Out from the Command Line

**Running LEF In from the GUI**

File ➔ Import ➔ LEF...
Design Data Translator's Reference
Translating LEF and DEF Files

Reads a LEF format library description file and stores information in the dfll library. While importing LEF, if the input LEF file size exceeds 2GB, following error message is displayed:

**Error* gec3LefInMain: Input File size exceeds 2GB. Cannot work with such size.

**Note:** Reading a LEF file does not delete cells that are already present. You can add pins to a cell by importing the version of the cell that includes pins. You can add ports to a pin if the pin does not have ports already defined for it, but you cannot overwrite existing ports. If you want to overwrite a cell, delete the cell before you use the File – Import – LEF command.

**LEF In Form**

**LEF File Name**

The LEF File Name field lets you specify a LEF format library description file that is to be read.

**Target Library Name**
The Target Library Name field lets you specify the dfII library created after importing LEF.

File – Import – LEF does not search reference libraries for vias or sites. Vias or sites not found in the target library are created in the target library.

**Overwrite Technology Information**

The Overwrite Technology Information option lets you overwrite the technology information in dfII. When this option is selected, information related to the following objects is overwritten in dfII:

- Use Min Spacing
- Clearance Measure
- Layer
- Via
- Via Rule
- Via Rule Generate
- Non Default Rule
- Same-Net Spacing Rules

The three situations in which the Overwrite Technology Information option would provide different results are described below:

1. If this option is selected (set to true) and the technology information is available, then the LEF information overwrites the technology file information.

   **Example**

   Consider a situation where layer metall in the LEF file has a width 0.500. Similarly, layer metall in the technology file has a width 0.750. After importing LEF, the width of the layer definition in the technology file changes to 0.500. This is because the selected Overwrite Technology Information option has overwritten the previous value 0.750 of the layer definition in the technology file.

2. If this option is selected (set to true) and the technology information is not available, then the following warning is displayed:

   Option Overwrite Technology Information is of no importance: Library does not exist.

3. If the option is not selected (not set to true) and the technology file information already exists, then the technology information is not overwritten.
Example

Consider a situation where layer metal1 in the LEF file has a width 0.500. Similarly, layer metal1 in the technology file has a width 0.750. After importing LEF, the width of the layer definition in the technology file remains unchanged. This is because the Overwrite Technology Information option is not selected. Due to this, values in the LEF file have not overwritten those in the technology file.

Use

The Use option lets you specify the libraries you want to include in the Ref. Library Names field for the cells not found in the target library.

Ref. Library Names

The Ref. Library Names field lets you specify the libraries File – Import – LEF will search if the Use option is selected. File – Import – LEF searches the libraries in the order you specify. The libraries you list must have identical technology information including layer, via, via rule, and site definitions. The libraries listed must not have different cells with the same cell name and view name.

MacroTarget View

The Macro Target View option lets you select the types of views to be created. The two options are abstract and layout. By default, abstract is selected.

If you select the abstract view, the layout view is not created for the macros.

View Name

The View Name field lets you specify the name of the view to be created. This field is disabled by default. It is enabled only after the macroTargetView value is changed to layout. The default value of the View Name field is abstract when the macroTargetView value is abstract and layout when the macroTargetView value is layout.

Delete Existing Connectivity in Macro CellView

The Delete Existing Connectivity in Macro CellView option lets you indicate if the existing connectivity in a macro cellview is to be deleted. This option is disabled by default and enabled only when the macroTargetView value is changed to layout. The default value is nil.

Target P&R Engine
The **Target P&R Engine** option indicates whether the LEF information being read is targeted for the Gate Ensemble or the Silicon Ensemble engine. By default, Silicon Ensemble is selected.

### Skip DB Locking

Use the **Skip DB Locking** option to enhance the speed of translation while importing LEF files. Select this option when no other process is accessing the target library in which you are translating the LEF file.

⚠️ **Caution**

*Do not use this option if multiple processes are accessing the target library.*

### Layer Map File Name

Use the **Layer Map File Name** field to specify a layer map file. A layer map file is an ASCII file that maps the layer names to the layer numbers in the technology library.

During LEF In, layers defined in the LEF file would be assigned a layer number as specified in the layer map file. For example, if the contents of the layer map file are:

```
M1 20
M2 40
```

then, layers **M1** and **M2** are assigned layer numbers **20** and **40** respectively in the technology library.

A layer map file must follow the conventions described below:

- It must have two fields, layer name and layer number, on every line. Use the following syntax to specify these two fields in a layer map file.

  ```
  layerName  layerNumber
  ```

- Every layer name defined would be created with corresponding layer number in the technology library.

  **Note:** The layer numbers defined in the layer map file should be in the CDB legal limit 0-194.

- Layer numbers must be unique.

  In a situation where the technology library already exists and you append the technology information from the LEF file to the technology library, ensure that the layers with the
same numbers as in the technology library do not exist. Incase of same layer numbers, the technology file compiler fails.

- Comments should be specified by putting # before a line.

**Note:** You can specify the layer numbers of as many layers as you require. For example, a LEF file could have 15 layers defined in it. Out of 15 layers, it is possible to have user-defined layer numbers for only 10 layers. The numbers of remaining five layers are generated by default.

### Running LEF In from the Command Line

You can import LEF files in the batch mode by running the `lefin` command. The syntax used to run LEF In is shown below:

```
lefin {-h | -help} 
   -lib libName
   [-refLib reflibs]
   [-engine engineName]
   [-view viewName]
   [-target targetViewType]
   [-deleteConnect]
   [-skipDbLock]
   [-overwrite]
   [-layerMap layerMapFile]}
```

where,

- `-h, -help` Print the help message.
- `-lef lefFileName` Input LEF file name.
- `-lib libName` Output library name.

If the output library does not exist, a library will be created in the current directory, which will also contain a technology database.

If the output library exists, it must contain a technology database, or should be attached to an already existing technology library.

- `-refLib reflibs` List of reference libraries.

You can specify multiple reference library names separated either by a space or a tab. LEF In searches for the reference libraries in the order in which you list the libraries.
## Design Data Translator’s Reference
### Translating LEF and DEF Files

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-engine engineName</td>
<td>Option to specify whether the cellview created is targeted for Gate Ensemble or Silicon Ensemble. You can specify either SE (for Silicon Ensemble) or GE (for Gate Ensemble). The default value is SE.</td>
</tr>
<tr>
<td>-view viewName</td>
<td>Output view name. You can update the output view name only when the target cellview is layout (-target layout). If the target cellview is abstract (-target abstract), the specified viewName is ignored.</td>
</tr>
<tr>
<td>-target targetViewType</td>
<td>Output target cellview type. You can specify either abstract or layout. The default target view type is abstract.</td>
</tr>
<tr>
<td>-deleteConnect</td>
<td>Option to delete the existing connectivity in macro cellview. This option lets you indicate if the existing connectivity in a macro cellview is to be deleted. This option works only when the cellview name is layout.</td>
</tr>
<tr>
<td>-skipDbLock</td>
<td>Enhance the speed of translation while importing LEF files. Set this option when no other process is accessing the target library in which you are translating the LEF file.</td>
</tr>
<tr>
<td>-overwrite</td>
<td>Overwrites the technology objects and macros. The default mode is update.</td>
</tr>
</tbody>
</table>
| -layerMap layerMapFile  | Input layer map file name. While creating layers, the layer numbers specified in this file are used. A layer map file contains a list of LEF layer names that map to the corresponding layer numbers. User-defined layer numbers can range from 0 through 194. The LEF layer names and layer numbers must be separated by a space. The entries in the layer map file are in the following format:  

```
layerName layerNumber
```

- **lefin** creates layer-purpose pairs, terminals on vias, and pin labels on pins. It also creates **cdsVia** corresponding to every **VIARULE GENERATE** statement in the LEF syntax. In addition, it creates the **display.drf** file that consists of packet display information. If **display.drf** already exists in the current file, the packet display
information is written in the display.drf.lefin file. Following warning message is generated in this situation.

*Warning* display.drf file exists. Writing the packet display information into file display.drf.lefin.

If the display.drf.lefin file exists in the current directory, it creates a new display.drf.lefin file by overwriting the previous one. Following warning message is generated in this situation.

*Warning* display.drf.lefin already exists. Overwriting it.

- The -appendTech option has been deleted. You can now import LEF files incrementally. Doing this does not change the definition of existing technology data, but if new data is defined in the LEF file, it is appended to the technology library.

- lefin creates new layer-purpose pairs and display packets to support the display of blockages and track patterns. It now creates display packets only for the following purposes: drawing, slot, gapfill, fill, boundary, net, pin, viaBlk, slotBlk, fillBlk, routeBlk, and track.

- During LEF In, via layers in the technology file now get populated.

- When any workbench is run, if display.drf does not exist in the directory from which it is run, the default display.drf file is loaded. During LEF In, if display.drf does not exist in the directory from which it is run, a new display.drf is created. After LEF In, the newly created display.drf is merged with the default one. On saving it, some extra sections, such as drDefineColor() and drDefineStipple(), are added to the display.drf file.

  If you do not save display.drf, the extra sections drDefineColor() and drDefineStipple(), are not added to the file.

Translating a Compressed LEF File

LEF In now provides the feature of translating the compressed LEF file. Supported compression utilities are gzip, bzip2, and compress. If a compressed file has .gz or .GZ extension, gzip utility is used to uncompress the file. If the file extension is .bz or .bz2, files are uncompressed using the bzip2 utility. Incase of .Z, files are uncompressed using the uncompress command.

Based on the file extension, LEF In assumes that it is a compressed file. If LEF In does not find the compression utility based upon the file extension, it displays the following fatal message:

Compressed File "file-name" open failed!!! PATH env variable not set for command "name_of_decompressor".
Running LEF Out from the GUI

File ➞ Export ➞ LEF...

Creates a LEF format library description file from a dfII library.

LEF Out Form

LEF File Name

The LEF File Name field lets you specify the name of the LEF file to be created.
Overwrite Existing LEF File?

The Overwrite Existing LEF File? option lets you overwrite the LEF file specified in the LEF File Name field.

Technology From Library

The Technology From Library option lets you specify the dfll library that contains the technology data, including layers, via rules, and site information. All the technology information you want to export to the LEF file must be present in this library.

Output Mode

The Output Mode option determines the kind of information to be included in the generated LEF file.

Logical & Physical

The Logical & Physical option generates a LEF file containing both the logical and physical information for the library cells.

While exporting LEF, you can choose to write or not write the technology information. To output a LEF file without technology information, set the SKILL variable as

gec3SkipLefTECH = t

To output a LEF file with technology information, set the SKILL variable as

gec3SkipLefTECH = 'unbound

Logical Only

The Logical Only option generates a LEF file containing only the logical (pin) information for the library cells.

Target P&R Engine

The Target P&R Engine option lets you specify the target engine for the generated LEF file.

Cell List File Name

The Cell List File Name field lets you specify the file name that contains the list of library cells to be converted in the LEF format. The default file name is ./lefout.list.

Use Layer List

The Use Layer List option is a boolean value. When this option is selected, the Layer list Name field is activated.
Layer List File Name

The *Layer List File Name* option lets you specify a file containing layer names which are to be exported in the output LEF file. Only these layers and the geometries associated with them are exported.

If you specify the absolute path of the file in the *Layer List File Name* field, the layers in the specified file are exported. If only the file name is specified, the file is searched in the current directory. You can separate layer names in the file by using a semi-colon, comma, space, or a new line character.

If the *Use Layer List* option is selected but no file is specified, the LEF Out translator quits the process and the following warning message is displayed:

*WARNING* Layer List Name is required.

If the file is not found, then the following error is displayed and LEF Out continues to ignore this option.

*WARNING* Layer List File <fileName> is not found or is not readable. Ignoring this option.

If the *Use Layer List* option is not selected, all the layers defined in the technology file are exported in the output LEF file.

Bus Bit Delimiter

Use the *Bus Bit Delimiter* field to set the bus bit characters in the output LEF file. The length of these characters must be exactly two. If you do not specify any value, by default, <> are accepted as bus bit characters in the output LEF file.

Hierarchy Delimiter

Use the *Hierarchy Delimiter* field to specify a character used to specify hierarchy in the output LEF file. To do this, enter only a single character in the field. If you do not specify any value, by default, | is accepted as a hierarchy delimiter in the output LEF file.

Compression

Use the *Compression* option to compress the output LEF file. Supported utilities for compressing files are gzip, bzip2, and compress.

- none
  - Use the *none* option if you do not want to compress the output LEF file.
- gzip
Design Data Translator's Reference
Translating LEF and DEF Files

Use the `gzip` utility to compress the output LEF file. If the output LEF file does not have .gz as the file extension, it adds .gz to the file name.

For example, the output LEF file `out.lef` changes to `out.lef.gz` but if the file name is `out.lef.gz`, then the file name remains unchanged. If `gzip` is not present in `PATH`, an uncompressed LEF file is generated and it has .lefdefTMP as the file extension. In addition, following warning message is generated.

"LEFOUT: Compressed File "file-name" open fail!!! PATH env variable not set for command "name_of_compressor", Opening as Plain ASCII file".

**bzip2**

Use the `bzip2` utility to compress the output LEF file. If the output LEF file does not have .bz2 as the file extension, it adds .bz2 to the file name.

For example, the output LEF file `out.lef` changes to `out.lef.bz2` but if the file name is `out.lef.bz2`, then the file name remains unchanged. If `bzip2` is not present in `PATH`, an uncompressed LEF file is generated and it has .lefdefTMP as the file extension. In addition, following warning message is generated.

"LEFOUT: Compressed File "file-name" open fail!!! PATH env variable not set for command "name_of_compressor", Opening as Plain ASCII file".

**compress**

Use the `compress` command to compress the output LEF file. If the output Stream file does not have .Z as the file extension, it adds .Z to the file name.

For example, the output LEF file `out.lef` changes to `out.lef.Z` but if the file name is `out.lef.Z`, then the file name remains unchanged. If `compress` is not present in `PATH`, an uncompressed LEF file is generated and it has .lefdefTMP as the file extension. In addition, following warning message is generated.

"LEFOUT: Compressed File "file-name" open fail!!! PATH env variable not set for command "name_of_compressor", Opening as Plain ASCII file".

Default: *none*

**Append Cell List File?**

The **Append Cell List File?** option specifies that the generated cell list appends to the existing cell list file specified in the **Cell List File Name** field.

**Overwrite Cell List File?**

The **Overwrite Cell List File?** option specifies that the generated cell list overwrites the file specified in the **Cell List File Name** field.

**Generate Cell List File By**
The *Generate Cell List File By* option specifies the cell list file generation utility for building the cell list. Choose one of the following utilities:

**Cells in Design**

The *Cells in Design* button lets you generate the cell list from all the master cells used in the design specified in the following form.

![Image of Generate Cell List File From Cells In Design dialog box]

**Pattern Match**

The *Pattern Match* button lets you generate a cell list by matching cell name patterns as specified in the following form. Specify the view name as `abstract`.

![Image of Find Cell Views By Pattern Match dialog box]
If the specified cell or view is not present in the library, the following warning message will be displayed:

"*WARNING* Cell list generated is empty. Put some valid LIB/CELL/VIEW names."

**Edit**

The *Edit* button displays a text window of the current cell list file. Use your text editor to edit the cell list file.

**Note:**

- To export LEF and DEF, you require the `prCellType` property on the abstract cellview. For details, see Appendix C, "CDB Abstract Cellview Properties for Exporting LEF/DEF".

- During LEF Out, if some of the cellviews are not found in the list of cellviews, the processing does not stop. Instead, all other valid cells having abstract or masklayout views are processed.

If LEF Out encounters the cellviews which are not present in the library or have its view other than abstract or masklayout, following warning message is displayed:

```
Cellview ("libName" "cellName" "viewName") is not an abstract maskLayout view
- will not be output to LEF
```

**Running LEF Out from the Command Line**

You can export LEF files in the batch mode by running the `lefout` command. The syntax used to run LEF Out is shown below:

```
lefout (-h | -lef lefFileName 
 -lib libName 
 -cellList cellListFileName 
 [-refLib reflibs] 
 [-engine engineName] 
 [-overwrite] 
 [-busBitChar char] 
 [-dividerChar char] 
 [-ver version] 
 [-compressionMode mode] 
 [-outputTechSection section] 
 [-outputMode mode] 
 [-layerMap layerMapFile])
```

where,

- `-h, -help` Print the help message.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-lef lefFileName</td>
<td>Output LEF file name.</td>
</tr>
<tr>
<td>-lib libName</td>
<td>Input library name.</td>
</tr>
<tr>
<td>-cellList cellListFileName</td>
<td>Input cell list file.</td>
</tr>
<tr>
<td></td>
<td>The entries in the cell list file should be in the following format:</td>
</tr>
<tr>
<td></td>
<td>ibName  cellName  viewName</td>
</tr>
<tr>
<td></td>
<td>where,</td>
</tr>
<tr>
<td></td>
<td>IbusName  Input library name.</td>
</tr>
<tr>
<td></td>
<td>cellName  Input cell name.</td>
</tr>
<tr>
<td></td>
<td>viewName  Input view name.</td>
</tr>
<tr>
<td></td>
<td>Though it is a mandatory argument but if you generate a technology-only LEF file (-outputMode logical), this argument is not needed.</td>
</tr>
<tr>
<td>-refLib reflibs</td>
<td>List of reference libraries.</td>
</tr>
<tr>
<td></td>
<td>You can specify multiple reference library names separated either by space or a tab. LEF Out searches for the reference libraries in the order in which you list the libraries.</td>
</tr>
<tr>
<td>-engine engineName</td>
<td>Option to specify whether the cellview created is targeted for Gate Ensemble or Silicon Ensemble.</td>
</tr>
<tr>
<td></td>
<td>You can specify either SE (for Silicon Ensemble) or GE (for Gate Ensemble). The default value is SE.</td>
</tr>
<tr>
<td>-overwrite</td>
<td>Overwrite the existing LEF file.</td>
</tr>
<tr>
<td>-busBitChar char</td>
<td>Set the bus bit characters in the output DEF file.</td>
</tr>
<tr>
<td></td>
<td>The length of these characters must be exactly two. The default value is &lt;&gt;.</td>
</tr>
<tr>
<td>-dividerChar char</td>
<td>Character to separate hierarchy in the output DEF file.</td>
</tr>
<tr>
<td></td>
<td>Specify only a single character for this option. The default value is</td>
</tr>
<tr>
<td>-ver version</td>
<td>Version number of the LEF/DEF format.</td>
</tr>
<tr>
<td></td>
<td>The default value is 5.4. You can specify a value up to 5.5.</td>
</tr>
<tr>
<td>-compressionMode mode</td>
<td>Compress the output DEF file by using the specified utility.</td>
</tr>
<tr>
<td></td>
<td>You can specify one from gzip, bzip2, compress, and none. The default value is none.</td>
</tr>
</tbody>
</table>
-outputTechSection

Option to specify the technology information to be printed to the LEF file.

You can specify either all, incremental, or none. The default value is all.

- If you set this option to all, complete technology information will be generated. So, the LEF file will contain both technology and macro information.
- If you set the option to incremental, only the site definitions along with the macro information will be printed in the LEF file. Layer, via definitions, and other technology data will not be output.
- If you set the option to none, no technology information will be generated. The LEF file would then contain only the macro information.

-outputMode mode

Option to specify the information to be included in the generated LEF file.

You can generate a LEF file with either only the technology information or both technology and macro information. To generate a technology-only LEF file, specify logical with the outputMode argument, otherwise specify logicalandphysical. The default value is logicalandphysical because by default LEF Out prints both technology and macro information in the output LEF file.

Important

If the outputMode is set to logical, the cellList argument is not needed.

-layerMap

Input layer map file name.

While creating layers, the layer numbers specified in this file are used. A layer map file contains a list of LEF layer names that map to the corresponding layer numbers. User-defined layer numbers can range from 0 through 194.

The LEF layer names and layer numbers must be separated by a space. The entries in the layer map file are in the following format:

layerName    layerNumber
Running DEF In from the GUI

File ➞ Import ➞ DEF...

Reads a DEF format design description into dfII. While importing DEF, if the input DEF file size exceeds 2GB, following error message is displayed:

*Error* gec3DEFInMain: Input File size exceeds 2GB. Cannot work with such size.

**Note:** When you read a DEF file, existing cells are not automatically deleted from the design. If you want to update a cell, delete the cell before you use the *File – Import – DEF* command. Otherwise, the design cellview will contain both new and old information.
**DEF In Form**

The *Library Name* field lets you specify an existing dfII library in which the DEF design is to be stored.

**Cell Name**

The *Cell Name* field lets you specify a cell name for the DEF design.
View Name

The View Name field lets you specify a view name for the DEF design. For example, if the design is not placed or routed, the view name is autoLayout.

Use

The Use field lets you specify the libraries you want to include in the Ref. Library Names field for the cells that are not present in the target library.

Import – DEF does not search reference libraries for vias or sites. Vias or sites not found in the target library are created in the target library.

Ref. Library Names

The Ref. Library Names field lets you specify the cell libraries to be searched, if the Use option is selected. File – Import – DEF searches the libraries sequentially. The libraries you specify must have identical technology information including layer, via, via rule, and site definitions. The libraries specified must have cells with unique cell names and view names.

Browse

The Browse button lets you select a library, cell, and view name from Library Manager.

DEF File Name

The DEF File Name field specifies the name of the DEF file to be read.

Map Names From

The Map Names From field lets you indicate the naming conventions used in the DEF file, so that the DEF net and instance names map correctly to dfII names.

VERILOG

The VERILOG option maps names originating from Verilog to dfII.

Startup Name Mapping

The Startup Name Mapping option maps DEF names to the valid names in dfII. Legal characters are different for dfII and LEF/DEF. Valid Characters explains how characters are mapped from DEF to dfII. When the Startup Name Mapping option is deselected, File – Import – DEF does not perform this mapping.

Note: Keep Startup Name Mapping deselected unless you want to read a design into dfII for the first time.
Target P&R Engine

The Target P&R Engine option specifies whether the cellview being created is targeted for Gate Ensemble or Silicon Ensemble. By default, Silicon Ensemble is selected.

Component Master Views

The Component Master Views field lets you specify a list of view names for the component master. The list is processed in the order of specification of the view names. The first available view of the master in the list is used to select the master.

Sections to Read

The Sections to Read option specifies which sections of the DEF file are read.

  All
  The All option reads entire sections. This option is selected by default.

  Nets
  The Nets option reads the Nets section, excluding the Special Nets section.

  Groups
  The Group option reads the Groups section.

  Constraints
  The Constraints option reads the Constraints section.

  Components
  The Components option reads the Components section.

  Note: File – Import – DEF does not support the component-based connectivity description. You must specify connectivity in the Nets section.

  Special Nets
  The Special Nets option reads the Special Nets section.

  Floorplan
  The FloorPlan option reads the Floorplan section.

  Note: When DEF In is invoked using the icca and icfb tools, the cellview has to be saved explicitly. If you invoke DEF In using the msfb, layout, or layoutPlus executables, the cellview is saved by default. If you do not want to save the cellview, set the SKILL variable
ldtrDonotSaveCellView to t. While saving the cellview, the following message is displayed:

Saving cell view <lib> <cell> <view>

where lib is the library name, cell is the cell name, and view is the view name of the cellview that is being saved.

Running DEF In from the Command Line

You can import DEF files in the batch mode by running the defin command. The syntax used to run DEF In is shown below:

```
defin {-h | -def defFileName
    -lib libName
    -cell cellName
    -view viewName
    [-refLib reflibs]
    [-engine engineName]
    [-deleteConnect]
    [-interactive]
    [-nameMap nameMapSource]
    [-sections DEFsections]
    [-vxldef]
    [-viewNameList viewNames]}
```

where,

- **-h, --help**
  Prints the help message.
- **-def defFileName**
  Input DEF file name.
- **-lib libName**
  Output library name.
  If the output library does not exist, a library will be created in the current directory, which will also contain a technology database.
  If the output library exists, it must contain a technology database, or refer to the technology database of another library.
- **-cell cellName**
  Output cell name.
- **-view viewName**
  Output view name.
-refLib reflibs
List of reference libraries.
You can specify multiple reference library names separated either by a space or a tab. DEF In searches for the reference libraries in the order in which you list the libraries.

-engine engineName
Target engine for the cellview created.
You can specify either SE (for Silicon Ensemble) or GE (for Gate Ensemble). The default value is SE.

-deleteConnect
Option to delete the existing connectivity in macro cellview.
This option lets you indicate if the existing connectivity in a macro cellview is to be deleted. This option works only when the cellview name is layout.

-interactive
Option to specify the execution of interactive DEF In. If this option is not specified, the design files are retrieved.
Interactive DEF In depends on the library property gec3NamesCaseSensitive. Retrieval of the design by DEF In is case sensitive.

-nameMap nameMapSource
Naming convention used in the DEF file to enable mapping of dfII net and instance names with DEF names.
You can specify one of VERILOG, VHDL, and DEF. The default value is VERILOG.

-sections DEFsections
List of sections to read from the DEF File.
The valid section names include NETS, SPECIALNETS, COMPONENTS, IOTIMINGS, GROUPS, FLOORPLAN, CONSTRAINTS. If you specify a section name other than these, it will be ignored by the translator.

-vxldef
Option to allow DEF In to map connectivity and name as per VXL requirements.

-viewNameList viewNames
List of view names for searching component masters.

**Translating a Compressed DEF File**

DEF In now provides the feature of translating the compressed DEF file. Supported compression utilities are gzip, bzip2, and compress. If a compressed file has .gz or .GZ extension, gzip utility is used to uncompress the file. If the file extension is .bz or .bz2, files
are uncompressed using the `bzip2` utility. In case of `.z`, files are uncompressed using the `uncompress` command.

Based on the file extension, DEF In assumes that it is a compressed file. If DEF In does not find the compression utility based upon the file extension, it displays the following fatal message:

Compressed File "file-name" open fail!!! PATH env variable not set for command "name_of_decompressor".

**Running DEF Out from the GUI**

File ➞ Export ➞ DEF...

Creates a DEF file from a dfll design cellview.

**Note:** If you have a hierarchical design, create the DEF file from the top-level cellview. Do not use File – Export – DEF on a subblock.
**Library Name**

The *Library Name* field lets you specify the name of the library that contains the cellview that is to be converted to the DEF format.

**Cell Name**

The *Cell Name* field lets you specify the cell that is to be converted to the DEF format.

**View Name**
The View Name field lets you specify the view that is to be converted to the DEF format.

Browse

The Browse button lets you select the library, cell, and view from the Library Manager.

DEF File Name

The DEF File Name field lets you specify the name of the DEF file to be created.

Map Names To

The Map Names To option lets you specify the naming conventions used in the DEF file, so that dfII net and instance names map correctly to DEF names.

VERILOG

The VERILOG option maps dfII names to DEF using Verilog naming conventions. This lets you recreate DEF names that originated from Verilog.

Overwrite existing DEF file

The Overwrite existing DEF file option lets you overwrite DEF file specified in the DEF File Name field.

If this option is deselected, and the file already exists, a message is displayed while exporting DEF and the translator stops.

Target P&R Engine

The Target P&R Engine option lets you select the Silicon Ensemble engine or the Gate Ensemble engine for which the DEF file is targeted. By default, Silicon Ensemble is selected.

Use Layer List

The Use Layer List option is a boolean value. When this option is selected, the Layer list File Name field is activated.

Layer List File Name

The Layer List File Name option lets you specify a file containing layer names which are to be exported in the output DEF file. Only these layers and the geometries associated with them are exported.

If you specify the absolute path of the file in the Layer List File Name field, the layers in the specified file are exported. If only the file name is specified, the file is searched in the current
directory. You can separate layer names in the file by using a semi-colon, comma, space, or a new line character.

If the *Use Layer List* option is selected but no file is specified, the DEF Out translator quits the process and the following warning is displayed:

*WARNING* Layer List Name is required.

If the file is not found, then the following error is displayed and DEF Out continues to ignore this option.

Layer List File <fileName> is not found or is not readable. Ignoring this option.

If the *Use Layer List* option is not selected, then all the layers defined in the technology file are exported in the output DEF file.

**Bus Bit Delimiter**

Use the *Bus Bit Delimiter* field to set the value of bus bit characters in the output DEF file. The length of these characters must be exactly two. If you do not specify any character, by default, <> are accepted as bus bit characters in the output DEF file.

**Hierarchy Delimiter**

Use the *Hierarchy Delimiter* field to specify a character used to express hierarchy in the output DEF file. To do this, enter only a single character in the field. If you do not specify any character, by default, | is accepted as a hierarchy delimiter in the output DEF file.

**Compression**

Use the *Compression* option to compress the output DEF file. Supported utilities for compressing files are gzip, bzip2, and compress.

- **none**
  
  Use the *none* option if you do not want to compress the output DEF file.

- **gzip**
  
  Use the *gzip* utility to compress the output DEF file. If the output DEF file does not have .gz as the file extension, it adds .gz to the file name.

  For example, the output DEF file out.def changes to out.def.gz but if the file name is out.def.gz, then the file name remains unchanged. If gzip is not present in PATH, an uncompressed DEF file is generated and it has .lefdetMP as the file extension. In addition, following warning message is generated.

  "DEFOUT: Compressed File "file-name" open failed!!! PATH env variable not set for command "name_of_compressor". Opening as Plain ASCII file".
**bzip2**

Use the `bzip2` utility to compress the output DEF file. If the output DEF file does not have `.bz2` as the file extension, it adds `.bz2` to the file name.

For example, the output DEF file `out.def` changes to `out.def.bz2` but if the file name is `out.def.bz2`, then the file name remains unchanged. If `bzip2` is not present in `PATH`, an uncompressed DEF file is generated and it has `.lefdefTMP` as the file extension. In addition, following warning message is generated.

"DEFOUT: Compressed File "file-name" open failed!!! PATH env variable not set for command "name_of_compressor". Opening as Plain ASCII file".

**compress**

Use the `compress` command to compress the output DEF file. If the output Stream file does not have `.Z` as the file extension, it adds `.Z` to the file name.

For example, the output DEF file `out.def` changes to `out.def.Z` but if the file name is `out.def.Z`, then the file name remains unchanged. If `compress` is not present in `PATH`, an uncompressed DEF file is generated and it has `.lefdefTMP` as the file extension. In addition, following warning message is generated.

"DEFOUT: Compressed File "file-name" open failed!!! PATH env variable not set for command "name_of_compressor". Opening as Plain ASCII file".

Default: `none`

---

**Running DEF Out from the Command Line**

You can export DEF files in the batch mode by running the `defout` command. The syntax used to run DEF Out is shown below:

```
defout (-h | -def defFileName
    -lib libName
    -cell cellName
    -view viewName
    [-engine engineName]
    [-nameMap nameMapSource]
    [-overwrite]
    [-vxldef]
    [-busBitChar char]
    [-dividerChar char]
    [-ver version]
    [-compressionMode mode]
    [-layerMap layerMapFile])
```
where,

- **-h, --help** Prints the help message.
- **-def defFileName** Output DEF file name.
- **-lib libName** Input library name.
- **-cell cellName** Input cell name.
- **-view viewName** Input view name.
- **-engine engineName** Target engine for the cellview created.
  You can specify either **SE** (for Silicon Ensemble) or **GE** (for Gate Ensemble). The default value is **SE**.
- **-nameMap nameMapSource** Naming convention used in the DEF file to enable mapping of dfII net and instance names with DEF names.
  You can specify one of **VERILOG**, **VHDL**, and **DEF**. The default value is **VERILOG**.
- **-overwrite** Overwrite the existing DEF file.
- **-vxldef** Option to allow DEF Out to map connectivity and name as per VXL requirements.
- **-busBitChar char** Set the bus bit characters in the output DEF file.
  The length of these characters must be exactly two. The default value is `<>`.
- **-dividerChar char** Character to separate hierarchy in the output DEF file.
  Specify only a single character for this option. The default value is `|`.
- **-ver version** Version number of the LEF/DEF format.
  The default value is 5.4. You can specify a value up to 5.5.
- **-compressionMode mode** Compress the output DEF file by using the specified utility.
  You can specify one from **gzip**, **bzip2**, **compress**, and **none**. The default value is **none**.
-layerMap  
layerMapFile

Input layer map file name.

While creating layers, the layer numbers specified in this file are used. A layer map file contains a list of LEF layer names that map to the corresponding layer numbers. User-defined layer numbers can range from 0 through 194.

The LEF layer names and layer numbers must be separated by a space. The entries in the layer map file are in the following format:

layerName    layerNumber
LEF Data Map

Silicon Ensemble uses LEF as the library implementation language to build the technology and macrocell library for a family of designs. The dfII database supports the representation of cell libraries and associated technology data in a different manner. The File – Import - LEF and File – Export – LEF commands provide a mechanism for moving library data between the two systems.

You can create a dfII library from one or more LEF files.

This topic describes the following LEF sections and shows how they map to dfII:

- **VERSION Statement** on page 324
- **UNITS Section** on page 324
- **AYER Section** on page 327
  - Routing Layer
  - Cut Layer
  - MASTERSLICE Layer
  - OVERLAP Layer
  - Implant Layer
- **ASTACK Statement** on page 354
- **PROPERTYDEFINITIONS Section** on page 355
- **IAS Section** on page 355
- **IARULE Section** on page 362
- **ONDEFALTRULE Section** on page 366
- **APING Section** on page 369
- **ITE Section** on page 369
- **ARRAYS Section** on page 370
- **ACRO Section** on page 372
- **IN Section** on page 376
- **BS Section** on page 384
Note: File – Import – LEF and File – Export – LEF do not support the PINPROPERTIES, MIN FEATURE, CROSS TALK, DIELECTRIC, IRDROP, or BEGINEXT statements.

VERSION Statement

The VERSION statement indicates the LEF version that was used to create a LEF file.

VERSION number ;

<table>
<thead>
<tr>
<th>LEF</th>
<th>dfII</th>
</tr>
</thead>
</table>

UNITS Section

The UNITS section describes the units of measurement used in the LEF file and the precision for each measure in database units. The values show how to interpret the numbers found in a LEF file.

[UNITS
    [ TIME NANOSECONDS convertFactor ; ]
    [ CAPACITANCE PICOFARADS convertFactor ; ]
    [ RESISTANCE OHMS convertFactor ; ]
    [ POWER MILLIWATTS convertFactor ; ]
    [ CURRENT MILLIAMPS convertFactor ; ]
    [ VOLTAGE VOLTS convertFactor ; ]}
You can set the value of `gec3DBUPerUU` in the following two ways:

- Setting the environment variable `gec3DBUPerUU`
- Setting the value of `DATABASE MICRONS` in the input LEF file

If you assign a value to `gec3DBUPerUU` by using both the above methods, the value specified in the LEF file takes precedence. If you do not use any of the above two methods, 100 is assigned as the default value to this variable.

The syntax of the `techParams` subsection in the technology file is shown below:

```plaintext
controls(
  techParams(
    ;(parameter value)
  )
)
```

In the above syntax, `parameter` refers to `LEFDEF_TIME_UNIT`, `LEFDEF_FREQUENCY_UNIT`, `LEFDEF_RESISTANCE_UNIT`, `LEFDEF_POWER_UNIT`, `LEFDEF_VOLTAGE_UNIT`, `LEFDEF_CURRENT_UNIT`, or `LEFDEF_CAPACITANCE_UNIT`.

```plaintext
[ DATABASE MICRONS convertFactor ; ]
[ FREQUENCY MEGAHERTZ convertFactor ; ]
END UNITS ]
```
Example

Consider an example that provides mapping between the statements in the UNITS section of the LEF file and the controls section of the technology file.

- **LEF File**

```
[UNITS
    [ TIME NANOSECONDS 1 ; ]
    [ FREQUENCY MEGAHERTZ 1 ; ]
END UNITS ]
```

- **Technology File**

```
controls(
    techParams(
        \{ LEFDEF_TIME_UNIT \ 1 \}
        \{ LEFDEF_FREQUENCY_UNIT \ 1 \}
    )
)
```

The only value you can change in the UNITS section is that of DATABASE MICRONS. The value of `convertFactor` is stored as the default value, 1, for all the units except DATABASE MICRONS. This value remains the same even if there is some other value specified in the input LEF file. If the current value is not equal to the default value, a warning is given to the user and the default value is stored in the technology file.

For DATABASE MICRONS, the permissible values of `convertFactor` are 100, 200, 1000, and 2000. The dfII database specifies this value in the .cdsenv file as DBUPerUU. The other values are fixed in the software application.

**Note:** The precision of the values of constructs is determined by the value you set for DATABASE MICRONS in the LEF file.

- If `convertFactor` is set to 200 or 1000, the precision is 3.
- If `convertFactor` is set to 2000, the precision is 4.
- In case of any value other than 200, 1000, or 2000, the precision is 2.

You can get twice the above mentioned precision by setting the value of the AREAPRECISION environment variable to NEW. When you use the AREAPRECISION environment variable, the high precision is applicable only to the AREA values such as AREA, MINENCLOSEDAREA, ANTEENAINPUTGATEAREA, ANTEENAINOUTDIFFAREA, ANTEENADIFFAREA, ANTEENAOOUTPUTDIFFAREA, ANTEENNAGATEAREA, ANTEENAPARTIALMETALAREA, ANTEENAPARTIALMETALSIDEAREA, and ANTEENAPARTIALCUTAREA. The other values appear with the normal precision governed by `gec3DBUPerUU`. For example, if you set the AREAPRECISION environment variable to NEW and `gec3DBUPerUU` to 2000, the AREA values will have a precision of 8.
LAYERS Section

The dfII database defines layers in the technology file. See the *Technology File and Display Resource File User Guide* for detailed information about defining layers in the technology file.

A LEF file can have the following types of layers:

- **Routing Layer**
- **Cut Layer**
- **MASTERSLICE Layer**
- **OVERLAP Layer**
- **Implant Layer**

The layer statements in a LEF file define the layer names and the associated place and route design rules with each routing layer.
Routing Layer

```
LAYER routingLayerName
    TYPE ROUTING ;
    ( PITCH distance ; )
    [ OFFSET distance ; ]
    [ MAXWIDTH width ;]
    [ PROTRUSIONWIDTH width1 LENGTH length WIDTH width2;]
    [ MINSTEP distance;]
    WIDTH defWidth ;
    [ AREA minArea ;]
    [ MINWIDTH width ;]
    [ MINIMUMCUT numCuts WIDTH width
      [ FROMABOVE|FROMBELOW]
      [ LENGTH length WITHIN distance];]
    [ MINENCLOSEDAREA area [ WIDTH width];]...
    [ SPACINGTABLE
      { PARALLELRUNLENGTH { length}
        [ WIDTH width { spacing}...]
        [ INFLUENCE
          [ WIDTH width WITHIN distance SPACING spacing}... ]
      ];...]
    [ SLOTWIREWIDTH minWireWidthForSlotting ;]
    [ SLOTWIRELENGTH minWireLengthForSlotting ;]
    [ SLOTWIDTH minSlotWidth ;]
    [ SLOTLENGTH minSlotLength ;]
    [ MAXADJACENTSLOTSPACING maxAdjSlotSpacing ;]
    [ MAXCOAXIALSLOTSPACING maxCoaxSlotSpacing ;]
    [ MAXEDGESLOTSPACING maxEdgeSlotSpacing ;]
    [ SPLITWIREWIDTH minWireWidthForSplitting ;]
    [ MINIMUMDENSITY minDensity ;]
    [ MAXIMUMDENSITY maxDensity ;]
    [ DENSITYCHECKWINDOW checkWindowLength checkWindowWidth ;]
    [ DENSITYCHECKSTEP checkStepValue ;]
    [ FILLACTIVESPACING fillToActiveSpacing ;]
    [ SPACING minSpacing
      [ RANGE minWidth maxWidth
        [ USELENGTHTHRESHOLD
          [ INFLUENCE infvalue
            [ RANGE stubMinWidth stubMaxWidth
              [ RANGE secMinWidth secMaxWidth
              ]
            ]
          ]
        ]
      ]
    ];
    { DIRECTION { HORIZONTAL | VERTICAL } ; }
    [ WIREEXTENSION value ; ]
    [ RESISTANCE RPERSQ value ; ]
    [ CAPACITANCE CPERSQDIST value ; ]
    [ HEIGHT distance ; ]
```
[ THICKNESS distance ; ]
[ SHRINKAGE distance ; ]
[ CAPMULTIPLIER value ; ]
[ EDGECAPACITANCE value ; ]
[ ANTENNAAREARATIO value ; ]
[ ANTENNAMODEL OXIDE {1,2,3,4};]
[ ANTENNAAREARATIO value ; ]
[ ANTENNADIFFAREARATIO { value | PWL ( ( d1 r1 ) ( d2 r2 ) ... ) } ; ]
[ ANTENNACUMAREARATIO value ; ]
[ ANTENNACUMDIFFAREARATIO { value | PWL ( ( d1 r1 ) ( d2 r2 ) ... ) } ; ]
[ ANTENNAAREAFACTOR value [ DIFFUSEONLY ] ; ]
[ ANTENNASIDEAREARATIO value ; ]
[ ANTENNADIFFSIDEAREARATIO { value | PWL ( ( d1 r1 ) ( d2 r2 ) ... ) } ; ]
[ ANTENNACUMSIDEAREARATIO value ; ]
[ ANTENNACUMDIFFSIDEAREARATIO { value | PWL ( ( d1 r1 ) ( d2 r2) ... ) } ; ]
[ ANTENNASIDEAREAFACTOR value [ DIFFUSEONLY ] ; ]
[ PROPERTY { propName propVal } ... ; ]
[ ACCURRENTDENSITY { PEAK | AVERAGE | RMS }]
{ value |
  FREQUENCY freq_1 freq_2 freq_3 ... freq_n ;
  [WIDTH width_1 width_2 width_3 ... width_m ; ]
  TABLEENTRIES
   v_freq_1_width_1 v_freq_1_width_2 ...
   v_freq_1_width_m
   v_freq_2_width_1 v_freq_2_width_2 ...
   v_freq_2_width_m
   ...
   v_freq_n_width_1 v_freq_n_width_2 ...
   v_freq_n_width_m
}
[ DCCURRENTDENSITY AVERAGE]
{ value |
  WIDTH width_1 width_2 ... width_m ;
  TABLEENTRIES value_1 value_2 ...value_m
}
END routingLayerName ;

The routing layer statements in the LEF file map to the technology file as described below:
The routing pitch of a routing layer is mapped to the `prRoutingPitch` subsection of the `prRules` section in the technology file.

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td>PITCH distance;</td>
<td><code>prRoutingPitch( (layerName Pitch))</code></td>
</tr>
</tbody>
</table>

The `Place and Route Rules` class sets the routing pitch for routing layers in the technology file.

- **LEF File**
  ```
  LAYER metall
  TYPE ROUTING;
  ....
  PITCH 2.4;
  ....
  END metall
  ....
  ```

- **Technology File**
  ```
  prRoutingPitch(
      (metall 2.4)
  ).
  ```

  ![Layername Pitch](image)

The routing `OFFSET` for a routing layer is mapped as `prRoutingOffset` in the `prRules` section of the technology file.

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFFSET distance;</td>
<td><code>prRoutingOffset( (layerName Offset))</code></td>
</tr>
</tbody>
</table>

The `Place and Route Rules` class sets the routing offset for routing layers in the technology file.

- **LEF File**
  ```
  LAYER metall
  TYPE ROUTING;
  ....
  ```
OFFSET 2.2;
...
END metal1
...

- Technology File

prRoutingOffset(
  (metal1 2.2)
) Layername Offset

- MAXWIDTH is mapped as maxWidth in the spacingRules section, which is a subsection of the physicalRules section in the technology file.

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXWIDTH width</td>
<td>(maxWidth layerName width)</td>
</tr>
</tbody>
</table>

The MAXWIDTH rule applies only for routing layers and is a float in units of um. If MAXWIDTH is specified in any other layer, following warning message will be displayed:

*WARNING* MAXWIDTH can be defined only in the routing layer: Ignoring the rule for layer layerName.

- PROTRUSIONWIDTH is mapped as protrusionWidth in the tableSpacingRules section, which is a subsection of the physicalRules section in the technology file.

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROTRUSIONWIDTH width1 LENGTH length WIDTH width2;</td>
<td>(&quot;protrusionWidth&quot; layer &quot;length&quot; nil nil &quot;width&quot; nil nil ((indices value)...))</td>
</tr>
</tbody>
</table>

where, width1, width2, and length are floats in units of um. The PROTRUSIONWIDTH rule applies only to the routing layer. A protrusion width must be greater than or equal to width1 if the protrusion connects to a wire having width greater than width2 and length greater than length.

- The MINSTEP rule specifies the minimum step size (or shortest edge length). In this rule, distance is a float in units of um and it specifies the minimum step size. The MINSTEP rule applies only to the routing layers.
MINSTEP is mapped as minStep in the spacingRules section, which is a subsection of the physicalRules section in the technology file.

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINSTEP distance; (minStep layerName distance)</td>
<td></td>
</tr>
</tbody>
</table>

- WIDTH is mapped as defaultWidth in the spacingRules section, which is a subsection of the physicalRules section in the technology file.

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIDTH defWidth; (defaultWidth layerName value)</td>
<td></td>
</tr>
</tbody>
</table>

- The AREA rule minimum area is mapped as minArea in the spacingRules section, which is a subsection of the physicalRules section in the technology file.

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td>AREA minArea; (minArea layerName minAreaValue)</td>
<td></td>
</tr>
</tbody>
</table>

- The MINWIDTH statement maps as minWidth in the spacingRules section, which is a subsection of the physicalRules section in the technology file.

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINWIDTH minWidth; (minWidth layerName value)</td>
<td></td>
</tr>
</tbody>
</table>

Note: If the values of defaultWidth and minWidth in the library that you export are equal, then MINWIDTH would be dropped in the exported LEF file and WIDTH would be exported.

- The MINIMUMCUT statement is mapped as minimumCuts in the spacingRules section, which is a subsection of the physicalRules section in the technology file. The FROMABOVE and FROMBELOW constructs are currently ignored because the technology file support does not exist for these constructs. During LEF In, when the FROMABOVE/ FROMBELOW construct is encountered, following warning message is generated:

*WARNING* Ignoring FROMABOVE/FROMBELOW MINIMUMCUT rules for layer layerName because they are not supported.
Similarly, during LEF In, the \texttt{LENGTH} construct is also ignored and if this construct is encountered, following warning message is generated:

\*WARNING\* Ignoring the \texttt{LENGTH MINIMUMCUT} rule for layer \texttt{layerName} because it is not supported.

\texttt{MINIMUMCUT} maps to the technology file as shown:

\begin{tabular}{|l|l|}
\hline
\textbf{LEF} & \textbf{Technology File} \\
\hline
\texttt{MINIMUMCUT numCuts} & (\texttt{minimumCut routingLayerName}) \\
\texttt{WIDTH width} & ("numCuts WIDTH minWidth") \\
\texttt{[FROMABOVE | FROMBELOW]} & \}
\texttt{[LENGTH length WITHIN distance]}; \\
\hline
\end{tabular}

where,

\begin{itemize}
\item \texttt{numCuts} \quad \text{Specifies the number of cuts required when the wire is greater than} \texttt{width}. \text{It is an integer without units.}
\item \texttt{FROMABOVE} \quad \text{Connections from the upper part of the layer}
\item \texttt{FROMBELOW} \quad \text{Connections from the lower part of the layer}
\item \texttt{length, width, and distance} \quad \text{Floats with units of} \texttt{um}
\item \texttt{LENGTH, WITHIN} \quad \text{Optional values which indicate that} \texttt{numCuts} \text{is required for thin wires directly connected to the wide wires if their width is greater than} \texttt{width} \text{and length greater than} \texttt{length}. \text{In addition,} \texttt{numCuts} \text{are also required for vias on the thin wire within the distance of the wide wire.}
\end{itemize}

\textbf{Example}

\begin{itemize}
\item \textbf{LEF File}
\begin{verbatim}
LAYER M3
  TYPE ROUTING ;
  WIDTH 3.60 ;
  AREA 24.50 ;
  MINIMUMCUT 2 WIDTH 2.0 ;
  ...
END M3
\end{verbatim}
\end{itemize}
- **Technology File**
  ```plaintext
  physicalRules(
    spacingRules(
      ;( rule layer1 layer2 value )
      ( minWidth M3 3.600 )
      ( minArea M3 24.500 )
      ( minimumCut M3 ("2.000 WIDTH 2.000")
      .......
      .......
    ) ; end of Spacing Rule
  )
  
  Note: LEF 5.5 supports more than one MINIMUMCUT rule per layer.
  ```

- In the MINENCLOSEDAREA construct, `area` specifies the minimum area size limit for metal that enclosed an empty area. It is a float in units of um^2. The width statement is optional and is a float in units of um. MINENCLOSEDAREA is mapped in technology file in two ways.

- When width statement is not specified
  ```plaintext
  In this case, MINENCLOSEDAREA is mapped as minEnclosedArea in the spacingRules section, which is the subsection of the physicalRules section in the technology file.
  ```

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINENCLOSEDAREA area;</td>
<td>(minEnclosedArea layerName area)</td>
</tr>
</tbody>
</table>

- When width statement is specified
  ```plaintext
  In this case, MINENCLOSEDAREA is mapped as minEnclosedArea in the tableSpacingRules section, which is a subsection of the physicalRules section in the technology file.
  ```

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINENCLOSEDAREA area [WIDTH width];</td>
<td>(&quot;minEnclosedArea&quot; layer &quot;width&quot; nil nil)((index value)...))</td>
</tr>
</tbody>
</table>

  Note: LEF 5.5 supports more than one value for the MINENCLOSEDAREA rule.
The **SPACINGTABLE** rule applies only to the routing layer. In this rule,

- **PARALLELRUNLENGTH** specifies that if the maximum width of the wire is greater than \( \text{width} \), and parallel run length greater than the \( \text{length} \), then wire spacing must be greater than or equal to \( \text{spacing} \).

- **INFLUENCE** specifies that if a wire width is greater than \( \text{width} \), and the distance to two other wires is less than \( \text{distance} \), then the other wires must be separated by distance greater than or equal to \( \text{spacing} \).

Where, \( \text{length} \), \( \text{width} \), and \( \text{spacing} \) are floats in units of \( \text{um} \).

**Example**

- **LEF File**

  ```
  LAYER M1
  TYPE ROUTING ;
  ....
  SPACINGTABLE
  PARALLELRUNLENGTH 0 .5 3 5
  WIDTH 0 0.15 0.15 .15 .1
  WIDTH .25 0.15 0.20 .20 .20
  WIDTH 1.5 0.15 0.50 .6 .6
  WIDTH 3 .15 .5 1 1
  WIDTH 5 .15 .5 1 2 ;
  ....
  END M1
  ```

- **Technology File**

  ```
  tableSpacingRules(
  ; ( rule layer1 [layer2]
  ; ( [index1Definitions] [index2Definitions]
  )
  )
  ("minSpacing" layer ("width" (widthindex1 widthindex2...)) "length"
  (lengthindex1 lengthindex2...))
  ((index value)...)
  ...);
  ) ;
  );
  ```
Design Data Translator's Reference
Translating LEF and DEF Files

; ( table ) )
("minSpacing" "M1" ("width" (0.00 0.25 1.50 3.00 5.00) nil "length"
(0.00 0.50 3.00 5.00) nil)
{
  ((5.00 ">") (5.00 ">")) 2.00
  ((3.00 ">") (3.00 ">")) 1.00
  ((1.50 ">") (3.00 ">")) 0.60
  ((1.50 ">") (0.50 ">")) 0.50
  ((0.25 ">") (0.50 ">")) 0.20
  ((0.00 ">") (0.00 ">")) 0.15
}
) ; end of minSpacing

❑ SPACINGTABLE INFLUENCE maps as minSpacingInfluence under the
tableSpacingRules subsection of the physicalRules section of the
technology file.

physicalRules(
  tableSpacingRules(
    ; ( rule              layer1        [layer2]
      ; ( [index1Definitions]       [index2Definitions]
        ; ( table ) )
    )
    ("minSpacing" layer ("width" (widthindex1 widthindex2...) "length"
      (lengthindex1 lengthindex2...))
      ((index value)...) )
    ("minSpacingInfluence" layer ("width" nil nil "distance" nil nil)
      ((index value)...) )
  )
) ;

Example

❑ LEF File

LAYER M4
  TYPE ROUTING ;
  DIRECTION VERTICAL ;
...
SPACINGTABLE
  PARALLELRUNLENGTH   0 .5
  WIDTH   0      0.15  0.15
  WIDTH .25  0.15  0.20 ;
SPACINGTABLE INFLUENCE
  WIDTH 1.5 WITHIN .5 SPACING .5
  WIDTH 3 WITHIN 1 SPACING 1
  WIDTH 5 WITHIN 2 SPACING 2 ;
....

END M4

❑ Technology File

; ( rule              layer1        [layer2]
  ; ( [index1Definitions]       [index2Definitions]
    ; ( table ) )
("minSpacing" "M4" ("width" (0.00 0.25) nil "length" (0.00 0.50) nil)
The metal slotting and splitting values SLOTWIREWIDTH, SLOTWIRELENGTH, SLOTWIDTH, SLOTLENGTH, MAXADJACENTSLOTSPACING, MAXCOAXIALSLOTSPACING, MAXEDGESLOTSPACING, and SPLITWIREWIDTH are written under the spacingRules subsection of the physicalRules section in the technology file.

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLOTWIREWIDTH minWireWidthForSlotting;</td>
<td>(slotWireWidth layerName minWireWidthForSlotting)</td>
</tr>
<tr>
<td>SLOTWIRELENGTH minWireLengthForSlotting;</td>
<td>(slotWireWidth layerName minWireLengthForSlotting)</td>
</tr>
<tr>
<td>SLOTWIDTH minSlotWidth;</td>
<td>(slotWidth layerName minSlotWidth)</td>
</tr>
<tr>
<td>SLOTLENGTH minSlotLength;</td>
<td>(slotLength layerName minSlotLength)</td>
</tr>
<tr>
<td>MAXADJACENTSLOTSPACING maxAdjSlotSpacing;</td>
<td>(maxAdjacentSlotSpacing layerName minAdjSlotSpacing)</td>
</tr>
<tr>
<td>MAXCOAXIALSLOTSPACING maxCoaxSlotSpacing;</td>
<td>(maxCoAxialSlotSpacing layerName maxCoaxSlotSpacing)</td>
</tr>
<tr>
<td>MAXEDGESLOTSPACING maxEdgeSlotSpacing;</td>
<td>(maxEdgeSlotSpacing layerName maxCoaxSlotSpacing)</td>
</tr>
</tbody>
</table>
Example

- **LEF File**

  ```
  LAYER M1
    TYPE ROUTING ;
    SLOTWIREFIREDTH 2.20 ;
    SLOTWIRELENGTH 3.10 ;
    SLOTWIDTH 0.50 ;
    SLOTLENGTH 4.50 ;
    MAXADJACENTSLOTSPACING 2.20 ;
    MAXCOAXIALSLOTSPACING 1.40 ;
    MAXEDGESLOTSPACING 1.10 ;
    SPLITWIREFIREDTH 0.20 ;
  ...
  END M1
  ```

- **Technology File**

  ```
  physicalRules(
    spacingRules(
      ...( rule layer1 layer2 value )
      ...
      ( slotWireWidth M1 2.200 )
      ( slotWireLength M1 3.100 )
      ( slotWidth M1 0.500 )
      ( slotLength M1 4.500 )
      ( maxAdjacentSlotSpacing M1 2.200 )
      ( maxCoaxialSlotSpacing M1 1.400 )
      ( maxEdgeSlotSpacing M1 1.100 )
      ( splitWireWidth M1 0.200 )
      ...
    ); end of Spacing Rule
  )
  ```

**Note:** The metal slotting values are written to the technology file only if the version is 5.4. For LEF versions lower or greater than 5.4, these values will not be written to the technology file.

- The metal filling values `MINIMUMDENSITY`, `MAXIMUMDENSITY`, `DENSITYCHECKWINDOW`, `DENSITYCHECKSTEP`, and `FILLACTIVESPACING` are written
under the `spacingRules` subsection of the `physicalRules` section in the technology file.

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td>MINIMUMDENSITY <code>minDensity</code>;</td>
<td><code>minimumDensity layerName minDensity</code></td>
</tr>
<tr>
<td>MAXIMUMDENSITY <code>maxDensity</code>;</td>
<td><code>maximumDensity layerName maxDensity</code></td>
</tr>
<tr>
<td>DENSITYCHECKWINDOW <code>checkWindowLength</code> <code>checkWindowWidth</code>;</td>
<td><code>densityCheckWindow layerName</code> <code>(&quot;checkWindowLength</code> <code>checkWindowWidth&quot;)</code></td>
</tr>
<tr>
<td>DENSITYCHECKSTEP <code>checkStepValue</code>;</td>
<td><code>densityCheckStep layerName</code> <code>checkStepValue</code></td>
</tr>
<tr>
<td>FILLACTIVESPACING <code>fillToActiveSpacing</code>;</td>
<td><code>fillActiveSpacing layerName</code> <code>fillToActiveSpacing</code></td>
</tr>
</tbody>
</table>

**Example**

- **LEF File**

  ```plaintext
  LAYER M2
  TYPE ROUTING ;
  ....
  MINIMUMDENSITY 75.0 ;
  MAXIMUMDENSITY 80.0 ;
  DENSITYCHECKWINDOW 2.0 2.30 ;
  DENSITYCHECKSTEP 2.20 ;
  FILLACTIVESPACING 2.20 ;
  ...
  END M2
  ```

- **Technology File**

  ```plaintext
  physicalRules(
  spacingRules(
    ;( rule layer1 layer2 value )
    ....
    ( minimumDensity M2 75.000 )
    ( maximumDensity M2 80.000 )
    ( densityCheckWindow M2 ("2.000 2.300" ))
    ( densityCheckStep M2 2.200 )
  )
  )
  ```
Spacing rules are mapped to the `spacingRules` section, which is a subsection of the `physicalRules` section in the technology file.

```
physicalRules(
    spacingRules(
        ;(rule layer1 [layer2] value)
    )
)
```

where,

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rule</code></td>
<td>Valid values are <code>minSpacing</code>, <code>minSpacingRange</code>, and <code>minSpacingThreshold</code>.</td>
</tr>
<tr>
<td><code>layer1</code></td>
<td>First layer name</td>
</tr>
<tr>
<td><code>layer2</code></td>
<td>Second layer name</td>
</tr>
<tr>
<td><code>value</code></td>
<td>Value is <code>minSpacing</code>. This value is a float.</td>
</tr>
</tbody>
</table>

- **SPACING** is mapped to the technology file as described below:

```
LEF                  Technology File
SPACING minspacing;  (minSpacing layerName
                        minSpacingRangevalue)
```

- **LENGTHTHRESHOLD** constructs in LEF are mapped to the technology file as described below:

```
LEF                  Technology File
SPACING minspacing  LENGTHTHRESHOLD
LENGTHTHRESHOLD
minSpacingThreshold
lengththresholdVal;
                        (minSpacingThreshold
                         lengththresholdVal
                         "minSpacing
                          LENGTHTHRESHOLD
                          lengththresholdVal")
```
RANGE constructs in LEF are mapped to the technology file as described below:

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPACING minSpacing LENGTHTHRESHOLD lengthThresholdVal RANGE minWidthVal maxWidthVal;</td>
<td>(minSpacingThreshold layerName “minSpacing lengthThresholdVal” lengthThresholdVal maxWidthVal)</td>
</tr>
<tr>
<td>SPACING minSpacing RANGE minWidth maxWidth;</td>
<td>(minSpacingRange “minSpacing Range minWidth maxWidth”)</td>
</tr>
<tr>
<td>SPACING minSpacing RANGE minWidth maxWidth USELENGTHTHRESHOLD;</td>
<td>(minSpacingRange “minSpacing Range minWidth maxWidth useLengthThreshold”)</td>
</tr>
<tr>
<td>SPACING minSpacing RANGE minWidth maxWidth INFLUENCE infValue;</td>
<td>(minSpacingRange “minSpacing Range minWidth maxWidth influence infValue”)</td>
</tr>
<tr>
<td>SPACING minSpacing RANGE minWidth maxWidth INFLUENCE infValue RANGE stubMinWidth stubMaxWidth;</td>
<td>(minSpacingRange “minSpacing Range minWidth maxWidth influence infValue Range stubMinWidth stubMaxWidth”)</td>
</tr>
<tr>
<td>SPACING minSpacing RANGE minWidth maxWidth RANGE secMinWidth secMaxWidth;</td>
<td>(minSpacingRange “minSpacing Range minWidth maxWidth Range secMinWidth secMaxWidth”)</td>
</tr>
</tbody>
</table>
Example

- **LEF File**

```plaintext
AYER M3
   TYPE ROUTING ;
   SPACING 1.60 ;
   SPACING 0.5 LENGTHTHRESHOLD 0.9 ;
   SPACING 0.5 LENGTHTHRESHOLD 1.0 RANGE 4.2 4.6 ;
   SPACING 0.8 RANGE 1.01 2000.0 USELENGTHTHRESHOLD ;
   SPACING 1.0 RANGE 10.50 ;
   SPACING 0.5 RANGE 10.0 1000.0 INFLUENCE 1.00 ;
   SPACING 0.5 RANGE 10.0 1000.0 INFLUENCE 1.0 RANGE .28 1.0 ;
   SPACING 0.5 RANGE 3.01 4.0 RANGE 4.01 5.0 ;
   ....
END M3
```

- **Technology File**

```plaintext
physicalRules(
   spacingRules(
      ;( rule layer1 layer2 value )
      ...
      ( minSpacing M3 1.600
      ( minSpacingThreshold M3 ( "0.500 LENGTHTHRESHOLD 0.900"
         "0.500 LENGTHTHRESHOLD 1.000 RANGE 4.200 4.600")
      ( minSpacingRange M3 ( "0.800 RANGE 1.010 2000.000 USELENGTHTHRESHOLD"
         "1.000 RANGE 10.000 50.000"
         "0.500 RANGE 10.000 1000.000 INFLUENCE 1.000"
         "0.500 RANGE 10.000 1000.000 INFLUENCE 1.000 RANGE 0.280 1.000"
         "0.500 RANGE 3.010 4.000 RANGE 4.010 5.000"
         "0.400 RANGE 3.010 4.000 RANGE 5.010 1000.0000" ))
```

The routing direction of the routing layers is mapped to the `prRoutingLayers` subsection of the `prRules` section in the technology file.

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
</table>
| DIRECTION {HORIZONTAL | VERTICAL}; | prRules(
| | prRoutingLayers(
| | ;(LayerName PreferredDirection) )}; |

Example
The **Place and Route Rules** class sets the routing direction for routing layers as described in the sample file below:

```plaintext
prRoutingLayers(
    ( metal1 "horizontal" )
    ( metal2 "vertical" )
    ( metal3 "horizontal" )
)
```

**Preferred directions**

Routing layers

- **Electrical rules of the routing layer** are mapped to the `characterizationRules` subsection in the technology file.

```plaintext
electricalRules(
    CharacterizationRules(
        ;(rule layer value)
    )
)
```

where,

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rule</code></td>
<td>Possible rules are <code>areaCap</code> (for <code>CAPACITANCE</code>), <code>edgeCapacitance</code> (for <code>EDGECAPACITANCE</code>), <code>sheetRes</code> (for <code>RESISTANCE</code>), <code>height</code> (for <code>HEIGHT</code>), <code>thickness</code> (for <code>THICKNESS</code>), <code>shrinkage</code> (for <code>SHRINKAGE</code>), <code>wireextension</code> (for <code>WIREEXTENSION</code>), or <code>capMultiplier</code> (for <code>CAPMULTIPLIER</code>).</td>
</tr>
<tr>
<td><code>layer</code></td>
<td>Layer name</td>
</tr>
<tr>
<td><code>value</code></td>
<td>Value of a rule</td>
</tr>
</tbody>
</table>

**Example**
The `characterizationRules` section of the `electricalRules` class associates electrical rules such as resistance and capacitance with routing layers as described below:

- **Area capacitance**
  ```plaintext
  characterizationRules(
    ( areaCap      metall  1.43-4 )
  )
 ```

- **Edge capacitance**
  ```plaintext
  characterizationRules(
    ( edgeCapacitance metall 0.6e-5 )
  )
 ```

- **Resistors**
  ```plaintext
  characterizationRules(
    ( sheetRes metall 0.040000 )
    ( height      metall 1.0 )
    ( thickness   metall 0.6 )
  )
  ```

- **Capacitance multiplier factor**
  ```plaintext
  characterizationRules(
    ( capMultiplier metall 1.0 )
  )
  ```

- **Antenna area factor**
  ```plaintext
  characterizationRules(
    ( antennaareafactor metall 0.0006
      ( antennaareafactor metall 0.0006
        ( antennaareafactor metall 0.0006
          ( antennaareafactor metall 0.0006
            ( antennaareafactor metall 0.0006
              ...)
        )
      )
    )
  )
  ```

Antenna rules in the routing layer are written in the technology file as characterization rules under the `electricalRules` section with a layer name and value.

```plaintext
electricalRules(
  characterizationRules(
    ;(rule layer value)
  )
)
```

Transistors with different oxide thickness or doping have different process antenna ratios. Therefore, an optional keyword `ANTENNAMODEL` has been added to the cut and routing layers. `ANTENNAMODEL` has four parameters: `OXIDE1`, `OXIDE2`, `OXIDE3`, or `OXIDE4`. The default parameter for `ANTENNAMODEL` is `OXIDE1`.

**Note:** Currently, only `OXIDE1` and `OXIDE2` are supported. `OXIDE3` and `OXIDE4` are ignored.

During translation, `ANTENNAMODEL` is mapped to the technology file as shown below:
# Oxide1 Mapping

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANTENNAMEODEL OXIDE1;</td>
<td>(defaultAntennaRule layer1</td>
</tr>
<tr>
<td>[ ANTENNAAREARATIO value ; ]</td>
<td>(AntennareaRatio</td>
</tr>
<tr>
<td>[ ANTENNADIFFAREARATIO { value</td>
<td>AntennasideAreaRatio</td>
</tr>
<tr>
<td></td>
<td>PWL ( ( d1 r1 ) ( d2 r2 ) ... )</td>
</tr>
<tr>
<td>} ;]</td>
<td>AntennadiffSideAreaRatio</td>
</tr>
<tr>
<td>[ ANTENNACUMAREARATIO value ; ]</td>
<td>AntennacumAreaRatio</td>
</tr>
<tr>
<td>[ ANTENNACUMDIFFAREARATIO</td>
<td>AntennacumDiffAreaRatio</td>
</tr>
<tr>
<td>{ value</td>
<td>PWL(( d1 r1)( d2 r2 ) ... ) } ;]</td>
</tr>
<tr>
<td>[ ANTENNAAREAFACTOR value</td>
<td>AntennaareaFactor</td>
</tr>
<tr>
<td>[ DIFFUSEONLY ] ; ]</td>
<td>AntennasideAreaFactor</td>
</tr>
<tr>
<td>[ANTENNASIDAREARATIO value ; ]</td>
<td></td>
</tr>
<tr>
<td>[ ANTENNADIFFSIDEAREARATIO</td>
<td></td>
</tr>
<tr>
<td>{ value</td>
<td>PWL (( d1 r1 ) ( d2 r2 ) ... ));]</td>
</tr>
<tr>
<td>[ANTENNACUMSIDEAREARATIO value ; ]</td>
<td></td>
</tr>
<tr>
<td>[ ANTENNACUMDIFFSIDEAREARATIO</td>
<td></td>
</tr>
<tr>
<td>{ value</td>
<td>PWL (( d1 r1 ) ( d2 r2) ...)});]</td>
</tr>
<tr>
<td>[ ANTENNASIDEAREAFACTOR value</td>
<td></td>
</tr>
<tr>
<td>[ DIFFUSEONLY ] ; ]</td>
<td></td>
</tr>
</tbody>
</table>
Oxide2 Mapping

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANTENNAMODEL OXIDE2;</td>
<td>(secondAntennaRule layer1</td>
</tr>
<tr>
<td>[ ANTENNAAREARATIO value ; ]</td>
<td>(AntennareaRatio</td>
</tr>
<tr>
<td>[ ANTENNADIFFAREARATIO { value AntennasideAreaRatio</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PWL ( ( d1 r1 ) ( d2 r2 ) ... AntennadiffAreaRatio</td>
</tr>
<tr>
<td>} ;]</td>
<td></td>
</tr>
<tr>
<td>[ ANTENNACUMAREARATIO value ;]</td>
<td>AntennacumAreaRatio</td>
</tr>
<tr>
<td>[ ANTENNACUMDIFFAREARATIO AntennacumDiffAreaRatio</td>
<td></td>
</tr>
<tr>
<td>{ value</td>
<td>PWL(( d1 r1)( d2 r2 ) AntennacumSideAreaRatio</td>
</tr>
<tr>
<td>... ) } ;] AntennacumDiffSideAreaRatio</td>
<td></td>
</tr>
<tr>
<td>[ ANTENNAAREAFACTOR value AntennaareaFactor</td>
<td></td>
</tr>
<tr>
<td>[ DIFFUSEONLY ] ; ] AntennasideAreaFactor</td>
<td></td>
</tr>
<tr>
<td>[ANTENNASIDEAREARATIO value ; ]</td>
<td></td>
</tr>
<tr>
<td>[ ANTENNADIFFSIDEAREARATIO</td>
<td></td>
</tr>
<tr>
<td>{ value</td>
<td>PWL ( ( d1 r1) ( d2 r2 ) ... )</td>
</tr>
<tr>
<td>[ANTENNACUMSIDEAREARATIO value</td>
<td></td>
</tr>
<tr>
<td>; ] AntennacumSideAreaRatio</td>
<td></td>
</tr>
<tr>
<td>[ ANTENNACUMDIFFSIDEAREARATIO AntennacumDiffSideAreaRatio</td>
<td></td>
</tr>
<tr>
<td>{ value</td>
<td>PWL ( ( d1 r1 ) ( d2 r2)...)};]</td>
</tr>
<tr>
<td>[ ANTENNASIDEAREAFACTOR value</td>
<td></td>
</tr>
<tr>
<td>[ DIFFUSEONLY ] ; ]</td>
<td></td>
</tr>
</tbody>
</table>

Example

- **LEF File**

```
LAYER M1
  TYPE ROUTING ;
  DIRECTION VERTICAL ;
  PITCH 5.6 ;
  WIDTH 2.6 ;
  SPACING 1.5 ;

# Oxide 1 values
ANTENNAMODEL OXIDE1 ;
ANTENNAAREARATIO 2.0 ;
ANTENNADIFFAREARATIO PWL ( (25 2.0 ) (35 3.0 ) ) ;
ANTENNACUMAREARATIO 2.1 ;
ANTENNACUMDIFFAREARATIO PWL ( (50 2.1 ) (60 3.2 ) (71 2.1 ) ) ;
ANTENNAAREAFACTOR 2.5 DIFFUSEONLY ;
ANTENNASIDEAREARATIO 2.1 ;
ANTENNADIFFSIDEAREARATIO PWL ( (20 1.0 ) (25 2.1 ) ) ;
```
ANTENNACUMSIDEAREARATIO 2.2 ;
ANTENNACUMDIFFSIDEAREARATIO PWL ( (10 2.1) (20 3.1) ) ;
ANTENNASIDEAREAFACOR 2.1 DIFFUSEONLY ;

# Oxide 2 values
ANTENNAMODEL OXIDE2 ;
ANTENNAAREARATIO 3.0 ;
ANTENNADIFFAREARATIO PWL ( (35 2.0) (35 3.0) ) ;
ANTENNACUMAREARATIO 3.1 ;
ANTENNACUMDIFFAREARATIO PWL ( (50 2.1) (60 3.2) (71 2.1) ) ;
ANTENNAAREAFACOR 3.5 DIFFUSEONLY ;
ANTENNASIDEAREARATIO 3.1 ;
ANTENNADIFFSIDEAREARATIO PWL ( (20 1.0) (25 2.1) ) ;
ANTENNACUMAREARATIO 3.2 ;
ANTENNACUMDIFFSIDEAREARATIO PWL ( (10 2.1) (20 3.1) ) ;
ANTENNASIDEAREAFACOR 3.1 DIFFUSEONLY ;

# Oxide 3 values
ANTENNAMODEL OXIDE3 ;
ANTENNAAREARATIO 4.0 ;
ANTENNASIDEAREAFACOR 4.1 DIFFUSEONLY ;

# Oxide 4 values
ANTENNAMODEL OXIDE4 ;
ANTENNASIDEAREAFACOR 5.1 DIFFUSEONLY ;
RESISTANCE RPERSQ 0.24 ; # Timing
CAPACITANCE CPERSQDIST 0.0006 ; # Timing
END M1
ANTENNACUMAREARATIO 4.1 ;

The AC and DC current density is mapped to the tableCharacterizationRules subsection of the electricalRules section in the technology file.

electricalrules(
   tableCharacterizationRules(
    (rule layer1 layer2)
    ;
    (index1Definitions [index2Definitions])
    ;
    )
    )
   )
)
where,

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>rule</td>
<td>ACCURRENTDENSITY PEAK, ACCURRENTDENSITY AVERAGE, ACCURRENTDENSITY RMS, and DCCURRENTDENSITY AVERAGE.</td>
</tr>
<tr>
<td>layer</td>
<td>Layer name</td>
</tr>
<tr>
<td>index1Definition</td>
<td>AC Current Density for Cut and Routing Layers</td>
</tr>
<tr>
<td></td>
<td>FREQUENCY ((f_1 f_2 \ldots f_m)) nil</td>
</tr>
<tr>
<td></td>
<td>DC Current Density for Routing Layer</td>
</tr>
<tr>
<td></td>
<td>FREQUENCY ((f_1 f_2 \ldots f_m)) nil</td>
</tr>
<tr>
<td></td>
<td>WIDTH ((w_1 w_2 \ldots w_m)) nil</td>
</tr>
<tr>
<td></td>
<td>DC Current Density for Cut Layer</td>
</tr>
<tr>
<td></td>
<td>FREQUENCY ((f_1 f_2 \ldots f_m)) nil</td>
</tr>
<tr>
<td></td>
<td>CUTAREA ((a_1 a_2 \ldots a_m)) nil</td>
</tr>
<tr>
<td>index2Definition</td>
<td>AC Current Density for Routing Layer</td>
</tr>
<tr>
<td>(optional)</td>
<td>WIDTH ((w_1 w_2 \ldots w_m)) nil</td>
</tr>
<tr>
<td></td>
<td>AC Current Density for Cut Layer</td>
</tr>
<tr>
<td></td>
<td>CUTAREA ((a_1 a_2 \ldots a_m)) nil</td>
</tr>
<tr>
<td></td>
<td>DC Current Density</td>
</tr>
<tr>
<td></td>
<td>nil</td>
</tr>
<tr>
<td>table</td>
<td>Value of the table depends on (index1Definition) and (index2Definition).</td>
</tr>
</tbody>
</table>

**Example**

Consider an example in which the AC and DC current density in routing layers is mapped to the `tableCharacterizationRules` subsection in the `electricalRules` section of the technology file.

- LEF File
Technological File

```
import LEF

while importing LEF, when the number of table entries is less than or more than the required
number, a warning message is displayed to the user. If the number of table entries is more
than required, the extra entries are ignored. Similarly if the table entries are less, a warning
given to the user.

In addition, if there are no values in the technology file for a particular Frequency-
Width or a Frequency-Cut area pair, then frequency is completely ignored and is not written
in the output LEF file. In addition, a warning is also given to the user.

Cut Layer

```

while exporting LEF, if there are no values in the technology file for a particular Frequency-
Width or a Frequency-Cut area pair, then frequency is completely ignored and is not written
in the output LEF file. In addition, a warning is also given to the user.
A new optional statement with `ADJACENTCUTS` and `WITHIN` has been added to the existing `SPACING` syntax in the cut layer. In the above syntax,

- `minSpacing` and `distance` are floats in units of um.
- `numCuts` is an integer that is either three or four.

If the `ADJACENTCUTS numCuts WITHIN` construct is specified, `minSpacing` applies only when the number of cuts are three or four specified within the `distance` spacing.

During translation, spacing rules are mapped to the technology file as:

```plaintext
tableSpacingRules(
  ...("viaSpacing" layer ("adjacentVias" nil nil "distance" nil nil)((index value))
  ...
)
```

Mapping of other rules is same for cut and routing layers. For details, see Routing Layer.

**MASTERSLICE Layer**

Masterslice layers are typically polysilicon layers and are only needed if the cell MACROs have pins on the polysilicon layer.

```plaintext
{ LAYER MastersliceLayerName
    TYPE { MASTERSLICE };
    END MastersliceLayerName } ...
```
Masterslice layer is mapped to the prMastersliceLayers subsection of the prRules section in the technology file.

```plaintext
prOverlapLayer(
    overlap  # Overlap layer
)
...
prMastersliceLayers(
    (diff   poly)  # List of masterslice layers
)
```

⚠️ **Important**

If you do not want a masterslice layer to appear in the output LEF file, you would need to update `techLayerProperties`, which is a subsection in the `layerDefinitions` section in the technology file. Add the `LEFaction` property to this subsection in the following format:

```plaintext
techLayerProperties(
    ;( PropertyName Layer1 [Layer 2] PropertyValue )
)
```

For example, with the following entry in the technology file:

```plaintext
techLayerProperties(
    ;( LEFaction NP “ignore” )
)
```

the ‘nplus’ layer NP will not appear in the output LEF file.

**Note:** Use the `LEFaction` property judiciously as it might have adverse impact on the performance of the translator.

**OVERLAP Layer**

In LEF, the overlap layer is used to store geometries that represent the prboundary. The overlap layer should normally be named OVERLAP.

```plaintext
{ LAYER OverlapLayerName
  TYPE { OVERLAP };
END OverlapLayerName } ...
```
The overlap layer is mapped to the `prOverlapLayer` subsection of the `prRules` section in the technology file.

```
prOverlapLayer(
    overlap
)
...
prMastersliceLayers(
    (diff poly)
)
```

**Implant Layer**

Some new technologies require high-threshold and low-threshold cells to be combined together. This has been made possible by creating the implant layer.

```
LAYER layerName
    TYPE IMPLANT;
    [WIDTH minWidth;]
    [SPACING minSpacing [LAYER layerName2];]...
    [PROPERTY propName propVal...;]...
END layerName
```

where,

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>layerName</code></td>
<td>Implant layer name</td>
</tr>
<tr>
<td><code>minWidth</code></td>
<td>Float in units of um</td>
</tr>
<tr>
<td><code>minSpacing</code></td>
<td>Float in units of um indicating the minimum spacing for the implant layer.</td>
</tr>
</tbody>
</table>

During translation, the layer name maps to the technology file as `layerName` in the `layerFunctions` section.

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>LAYER layerName;</code></td>
<td><code>(layerName &quot;nplus&quot;)</code></td>
</tr>
</tbody>
</table>
Width is mapped as `minWidth` in the `spacingRules` section, which is a subsection of the `physicalRules` section of the technology file. Similarly, spacing is also as `minSpacing` mapped in the `spacingRules` section.

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAYER <code>layerName</code>;</td>
<td><code>(layerName &quot;nplus&quot;)</code></td>
</tr>
<tr>
<td>SPACING <code>minSpacing</code></td>
<td><code>(minSpacing layerName value)</code></td>
</tr>
<tr>
<td><code>[LAYER layerName2];</code></td>
<td></td>
</tr>
</tbody>
</table>

**Example**

**LEF File**

```
LAYER IMPLANT1
    TYPE IMPLANT ;
    WIDTH 1.0 ;
    SPACING 2.3 ;
END IMPLANT1
```

**Technology File**

```
layerFunctions(
    ;( Layer Functions )
    ( IMPLANT1     "nplus" )
    ( IMPLANT2     "nplus" )
    ( IMPLANT3     "nplus" )
) ; End of layerFunctions.
physicalRules(
    spacingRules(  
    ( minWidth    IMPLANT1 1.000 )
    ....
    ( minSpacing  IMPLANT1 2.3000)
    ....
  ))
```

**Important**

If you do not want an implant layer to appear in the output LEF file, you would need to update `techLayerProperties`, which is a subsection in the `layerDefinitions` section in the technology file. Add the `LEFaction` property to this subsection in the following format:

```
technLayerProperties(
    ;( PropertyName Layer1 [Layer 2] PropertyValue )
)
```
For example, with the following entry in the technology file:

```
techLayerProperties(
   ;( LEFaction NP "ignore" )
)
```

the ‘nplus’ layer NP will not appear in the output LEF file.

**Note:** Use the **LEFaction** property judiciously as it might have adverse impact on the performance of the translator.

### MAXVIASTACK Statement

Some processes have an overall limit on the number of single-cut stacked vias that are allowed on top of each other. The **MAXVIASTACK** keyword has been added to limit the number of stacked vias. Currently, four stacked vias are permitted on top of each other.

```
[MAXVIASTACK value [ RANGE bottomLayer topLayer];]
```

where,

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>An integer that limits the number of single stacked vias that are allowed on top of each other.</td>
</tr>
<tr>
<td><code>bottomLayer</code></td>
<td>Routing layer names</td>
</tr>
<tr>
<td><code>topLayer</code></td>
<td></td>
</tr>
</tbody>
</table>

**MAXVIASTACK** is mapped in the technology file in the **prMaxStackVias** section, which is a subsection of the **prRules** section.

### Syntax Description

- **value:** An integer that limits the number of single stacked vias that are allowed on top of each other.
- **bottomLayer** and **topLayer:** Routing layer names

If the specified layer names are not routing layer names, the **RANGE** keyword is ignored and following warning message is displayed:

```
*WARNING* The layer names bottomLayer, topLayer for MAXVIASTACK are not routing layer names: Ignoring the RANGE keyword.
```

If you specify more than one **MAXVIASTACK** keywords, only the last keyword is written and others are ignored. Following warning message is displayed:
*WARNING* Only one MAXVIASTACK keyword is allowed, the last MAXVIASTACK is retained.

The MAXVIASTACK should follow the layer definitions in the LEF file.

**PROPERTYDEFINITIONS** Section

The PROPERTYDEFINITIONS section of the LEF file lists the properties you can provide to libraries, pins, macros, and vias.

```
PROPERTYDEFINITIONS
    objectType propName propType [RANGE # #] [value | stringValue]; ...
END PROPERTYDEFINITIONS
```

File – Import – LEF stores the properties listed in the LEF file as hierarchical properties of the library. File – Export – LEF uses the dfII library properties to recreate the PROPERTYDEFINITIONS section.

Example

```
PROPERTYDEFINITIONS
    LIBRARY versionNum INTEGER 12;
    LIBRARY title STRING "Cadence96";
    VIA count INTEGER RANGE 1 100;
    MACRO weight REAL RANGE 1.0 100.0;
    MACRO type STRING;
END PROPERTYDEFINITIONS
```

**VIAS Section**

LEF defines the vias for non-default regular wiring.

```
{ VIA viaName [ DEFAULT ]
    [ TOPOFSTACKONLY ]
    [ FOREIGN foreignCellName
        [ pt [ orient ] ] ]
    { LAYER layerName ; [ RECT pt pt ; ]...}...
    [ RESISTANCE resistanceValue ; ]
    [ SYMMETRY orientation ; ]
    [ PROPERTY { propName propVal}...; ]...
END viaName }...
```
The via section is mapped in dfII as described below:

<table>
<thead>
<tr>
<th>LEF</th>
<th>Technology File</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIA vianame</td>
<td>Cell name of via cellview</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>Via has the default via type. The dfII database defines default vias with symmetrical enclosures on all layers as symbolic contact devices. The dfII database defines nondefault vias and asymmetrical default vias as rule contact devices.</td>
</tr>
<tr>
<td>TOPOFSTACKONLY</td>
<td>Via has the topstack property</td>
</tr>
<tr>
<td>LAYER layerName...</td>
<td>Cut layer and metal (or masterslice) layers of via cellview.</td>
</tr>
<tr>
<td>RECT pt pt</td>
<td>Rectangular geometry on via layer</td>
</tr>
<tr>
<td>FOREIGN FOREIGN</td>
<td>gec3Foreign property of the via cellview</td>
</tr>
<tr>
<td>FOREIGN foreignCellName</td>
<td></td>
</tr>
<tr>
<td>RESISTANCE resistanceValue</td>
<td>res property of the via cellview</td>
</tr>
<tr>
<td>PROPERTY propName propVal</td>
<td>Property of the via cellview</td>
</tr>
</tbody>
</table>

The dfII database defines default vias as symbolic contact devices in the Device Definitions class of the technology file. Default vias can also be defined as user-defined devices in the Device Definitions class of the dfII technology file.

**Important**

If a technology file contains vias defined on layers that are not defined in the technology file, set the environment variable DROP_VIAS_ON_UNDEF_LAYERS. This environment variable ensures that the vias on undefined layers are not output in the LEF file.

Vias are mapped to three sections of the technology file:

- symContactDevice
- ruleContactDevice
- cdsViaDevice

These sections are described below:
symContactDevice

All the default vias defined in the LEF file are mapped to this section of the technology file.
Example

- **LEF File**
  ```
  VIA VIA12 DEFAULT
  RESISTANCE 0.0500000000 ;
  LAYER M1 ;
  RECT -2.00 -2.00 2.00 2.00 ;
  LAYER CUT12 ;
  RECT -0.70 -0.70 0.70 0.70 ;
  LAYER M2 ;
  RECT -2.00 -2.00 2.00 2.00 ;
  END VIA12
  ```

- **Technology File**
  ```
  symContactDevice{
  ;(name viaLayer viaPurpose layer1 purpose1 layer2 purpose2
  ;w l(row column xPitch yPitch xBias yBias)encByLayer1 encByLayer2
  legalRegion)
  "VIA12" "CUT12" drawing "M1" drawing "M2" drawing
  1.400 1.400 (1 1 1.000 1.000 center center) 1.300 1.300 _NA_)

  ruleContactDevice
  ```

All vias without the DEFAULT keyword are mapped to this section in the technology file.

Example

- **LEF File**
  ```
  VIA VIABIGPOWER12
  RESISTANCE 0.2000000000 ;
  LAYER M1 ;
  RECT -21.000 -21.000 21.000 21.000 ;
  LAYER CUT12 ;
  RECT -2.400 -0.800 2.400 0.800 ;
  RECT -19.000 -19.000 -14.200 -17.400 ;
  RECT -19.000 17.400 -14.200 19.000 ;
  RECT 14.200 -19.000 19.000 -17.400 ;
  RECT 14.200 17.400 19.000 19.000 ;
  RECT -19.000 -0.800 -14.200 0.800 ;
  RECT -2.400 -19.000 2.400 -17.400 ;
  RECT 14.200 -0.800 19.000 0.800 ;
  RECT -2.400 17.400 2.400 19.000 ;

  LAYER M2 ;
  RECT -21.000 -21.000 21.000 21.000 ;
  END VIABIGPOWER12
  ```

- **Technology File**
  ```
  ruleContactDevice(
  ```
cdsViaDevice

All asymmetrical default vias are mapped to this section of the technology file. Up to 5.0.32 release, the asymmetric vias mapped to the symEnhContactDevice section of the technology file.

When File – Import – LEF encounters asymmetrical default vias (that is, default vias with unequal overlaps in either x or y directions or both in the LEF file), it creates cdsVia devices for the default vias in the dfII design.

When File – Export – LEF encounters a default symEnhContact or cdsVia device in the dfII design, it writes it as a default via with rectangles on the various layers in the LEF file.

Example

- LEF File

```lef
VIA M2_M1_EXT_RIGHT DEFAULT
   LAYER metal1 ;
   RECT -0.250 -0.250 0.250 0.250 ;
   LAYER v1 ;
   RECT -0.200 -0.200 0.200 0.200 ;
   LAYER metal2 ;
   RECT -0.350 -0.350 0.650 0.350 ;
END M2_M1_EXT_RIGHT
```

Here, the value 0.650 in metal2 layer causes this via to become asymmetric.

- Technology File

Information related to M2_M1_EXT_RIGHT via is mapped in two subsections of the technology file.
As `minEnclosure` rule entry in the `orderedSpacingRules` subsection of the `physicalRules` section of the technology file.

```plaintext
designRules(
    ...
    orderedSpacingRules(
        ;( rule    layer1        layer2        value )
        ( minEnclosure   "metal1"       "v1"       0.050 )
        ( minEnclosure   "metal2"       "v1"       0.150 )
        ...)
    ); end of orderedSpacingRules
    ...)
); end of Physical Rules
```

As device parameter entry within the `cdsViaDevice` subsection of the `devices` section of the technology file.

```plaintext
designs(
    ...
    cdsViaDevice(
        ;( name cutLayer cutPurpose layer1 purpose1 layer2 purpose2
        ; row column origin stackedVias cutLayerW cutLayerL xCutSpacing yCutSpacing
        ; layer1XDirOverride layer1YDirOverride layer2XDirOverride layer2YDirOverride
        ; layer1Dir layer2Dir
        ; layer1XDefOverride layer1YDefOverride layer2XDefOverride layer2YDefOverride
        ; implantLayer1 implantLayer2 diffSpacing abutClass)
        ("M2_M1_EXT_RIGHT" "v1" drawing "metal1" drawing "metal2" drawing 1 1
centerCenter t 0.400 0.400 0.000 0.000 _NA_ _NA_ 0.450 _NA_ "r"
    _NA_ _NA_ _NA_ _NA_ nil nil _NA_ ""
        )
    ); end of cdsViaDevice
```

**Important**

cdsVia devices are sensitive to the manufacturing grid values. If manufacturing grid is not specified in the LEF file, then by default, it is assumed as 0.10 during LEFIn (technology library creation). To get the correct rectangle dimensions, specify some compatible value.

**Opus Symbolic Device Declaration** - You might change layer and device pcell names and contact parameter values to fit your library. In the above example, `diff` is used for diffusion layer, `cont` for contact layer, `metal1` for first metal, and `via` for an additional contact layer. The default enclosure by `diff` is 0.3 (`encByLayer1`), and that by `metal1` is 0.2 (`encByLayer2`). The `diff` enclosure determined by `layer1Dir` is 0.1 (`layer1XEnc`) for horizontal direction and 0.2 for vertical direction (`layer1YEnc`).

Similarly, `layer2XEnc` and `layer2YEnc` controlled by `layer2Dir` specifies that the horizontal and vertical enclosures for `metal1` are 0.0 and 0.4, respectively. Valid values for `layer1Dir` and `layer2Dir` are:

```
"x", "xy", "xt", "xb", "y", "yx", "y1", "yr" and "".
```

x=horizontal, y=vertical, t=top, b=bottom, l=left, r=right
The first character in layer1Dir and layer2Dir determines layer enclosure, and it also sets the pin access direction for pins of the layer inside the contact. The second character in layer1Dir and layer2Dir adds additional direction(s) for pin access. The double quotation marks sets the default enclosure and pin access direction to all the four sides.

In order to specify different overlaps in the x and y directions using the syEnhContact device, the layer1Dir and layer2Dir parameters must be used to specify the enclosures for layer1 and layer2. If layer1 is "", the encByLayer1 is used for both x and y enclosures of layer1, and the layer1XEnc and layer1YEnc are ignored. If layer1Dir is xy, layer1XEnc is used for enclosure in the x direction while encByLayer1 is used for enclosure in the y direction. Pins can be accessed in both x and y directions on layer1.

If layer1Dir is yx, layer1YEnc is used for enclosure in the y direction while encByLayer1 is used for enclosure in the x direction. Pins can be accessed in both y and x directions on layer1. It works similarly for layer2.

Default vias also are assigned the default via type in the via types section of the Place and Route Rules class as shown below:

```plaintext
prViaTypes(
  ( M2_M1 symbolic) "default"
.. Via cell name View name Via type
```


Via layers for all the vias defined in the LEF file are mapped to the viaLayers subsection of the layerRules section in the technology file as shown below.

```plaintext
layerRules(
viaLayers(
 ;( layer1 viaLayer layer2 )
  ( PW CUT01 M1 )
  ( NW CUT01 M1 )
  ( PD CUT01 M1 )
  ( ND CUT01 M1 )
  ( POLYS CUT01 M1 )
  ( M1 CUT12 M2 )
  ( M2 CUT23 M3 )
); end of viaLayers
); end of layerRules
```
Via properties are mapped to \texttt{tfcDefineDeviceProp} subsection of the \texttt{devices} section in the technology file as shown below:

\begin{verbatim}
tfcDefineDeviceProp(
    ; (viewName deviceName propName propValue)
    (symbolic "C2PW"        res     1.900000E+01)
    (symbolic "C2NW"        res     2.100000E+01)
    (symbolic "IN1X-via"    topstack t)
    (symbolic "IN1X-via"    gec3Foreign "IN1X")
)
\end{verbatim}

**VIARULE Section**

The \texttt{VIARULE} section in the LEF file specifies vias to be used at the intersection of special wires of the same net. A via rule can refer to vias that are defined elsewhere in the LEF file or generate its own vias.

**Listing Predefined Vias**

The following LEF syntax specifies a via rule using predefined vias:

\begin{verbatim}
VIARULE viaRuleName
  LAYER layerName;
  DIRECTION {HORIZONTAL | VERTICAL} ;
  [WIDTH minWidth TO maxWidth ;]
  [OVERHANG minOverhang ;]
  [METALOVERHANG maxMetalOverhang ;]
  LAYER layerName;
  DIRECTION {HORIZONTAL | VERTICAL} ;
  [WIDTH minWidth TO maxWidth ;]
  [OVERHANG minOverhang ;]
  [METALOVERHANG maxMetalOverhang ;]
  {VIA viaName ;}...
END viaRuleName
\end{verbatim}

The via rules are mapped in the \texttt{prViaRules} section of the technology file as shown below:

\begin{verbatim}
prViaRules(
    ; viaRuleName viaDeviceNameList
    ; layer1 direction (wMin wMax overHang metalOverHang)
    ; layer2 direction (wMin wMax overHang metalOverHang)
    ...
)
\end{verbatim}

**Example**
LEF File

VIARULE VIASP21
  LAYER METAL1;
  DIRECTION VERTICAL;
  WIDTH .6 TO 1.8;
  METALOVERHANG .6;
  LAYER METAL2;
  DIRECTION HORIZONTAL;
  WIDTH .6 TO 1.8;
  METALOVERHANG .6;
  VIA CENTER21;
  VIA TOP21;
  VIA BOTTOM21;
END VIASP21

Technology File

The dfIl technology file specifies this type of via rule in the prViaRules section of the Place and Route Rules class.

prViaRules(
  (“viaSP21” (CENTER21 TOP21 BOTTOM21)
    (“vertical” (6 1.8 _NA_ .6)
      Minimum width
      Maximum width
    (“horizontal” (6 1.8 _NA_ .6)
      Minimum overhang
      Maximum metal overhang
  )
)

List of vias. Via definitions are in the Devices class.
Generating Vias

In LEF, a via rule using generated vias has the following syntax:

```
VIARULE viaRuleName GENERATE
   LAYER layerName;
   {DIRECTION {HORIZONTAL | VERTICAL};
    [WIDTH minWidth TO maxWidth;]
    [OVERHANG minOverhang;]
    [METALOVERHANG maxMetalOverhang;]
    |ENCLOSURE overhang1 overhang2;}
   LAYER layerName;
   {DIRECTION {HORIZONTAL | VERTICAL};
    [WIDTH minWidth TO maxWidth;]
    [OVERHANG minOverhang;]
    [METALOVERHANG maxMetalOverhang;]
    |ENCLOSURE overhang1 overhang2;}
   LAYER layerName;
   LAYER layerName;
   RECT pt pt;
   SPACING xSpacing BY ySpacing;
   [RESISTANCE resistancePerCut;]
END viaRuleName
```

In LEF 5.5, via rule generate supports the ENCLOSURE rule. The ENCLOSURE rule provides that a via must be covered by metal on two opposite sides by overhang1 and two other sides by overhang2. Overhang1 and overhang2 are floats with units of um.

If you specify the ENCLOSURE statement incorrectly, various warning messages are generated.

- If the ENCLOSURE statement is specified after DIRECTION, following warning message is generated:
  
  `*WARNING* The keyword DIRECTION already specified for the viaRule viaRuleName, layer layerName. Ignoring DIRECTION keyword.`

- If DIRECTION is specified after ENCLOSURE, following warning message is generated:
  
  `*WARNING* The keyword ENCLOSURE already specified for the viaRule viaRuleName, layer layerName. Ignoring DIRECTION keyword.`

- If keywords OVERHANG and METALOVERHANG are specified with the ENCLOSURE statement, following warning messages are generated:
  
  `*WARNING* The keyword ENCLOSURE already specified for the viaRule viaRuleName, layer layerName. Keyword OVERHANG not allowed with ENCLOSURE. Ignoring OVERHANG keyword.`

  `*WARNING* The keyword ENCLOSURE already specified for the viaRule viaRuleName, layer layerName. Keyword METALOVERHANG not allowed with ENCLOSURE. Ignoring METALOVERHANG keyword.`
The generating vias section is mapped to the prGenViaRules section of the technology file.

```plaintext
prGenViaRules(
    ; viaRuleName viaLayer (wl xPitch yPitch Res)
    ; layer1 direction (overhang1 overhang2) (wMin wMax overHang metalOverHang)
    ; layer2 direction (overhang1 overhang2) (wMin wMax overHang metalOverHang)
    ....
)
```

**Example**

- **LEF File**

```
VIARULE VIAGEN21 GENERATE
LAYER METAL1;
    DIRECTION HORIZONTAL;
    WIDTH .6 TO 20.0;
    OVERHANG .6;
    METALOVERHANG .6;
LAYER METAL2;
    DIRECTION VERTICAL;
    WIDTH .6 TO 20.0;
    OVERHANG .6;
    METALOVERHANG .6;
LAYER VIA
    RECT .6 .6;
    SPACING 1.2 BY 1.2;
END VIAGEN21
```

- **Technology File**

After mapping to the Place and Route Rules class of the dfII technology file, it appears as shown below:

```
prGenViaRules(
    (viagen21via (.6 .6 1.2 1.2 _NA_ )
    metal1“horizontal” (.6 20.0 .6.6 )
    metal2“vertical” (.6 20.0 .6.6 )
)
```
Turn Via Rules

The LEF file also uses the VIARULE statement to specify turn vias. Turn via rules fill the corner area on the same layer at the intersection between two special wires. The following example allows turn vias on layer M1:

**LEF File**

VIARULE TURN1 GENERATE
  LAYER M1 ;
  DIRECTION VERTICAL ;
  LAYER M1 ;
  DIRECTION HORIZONTAL ;
END TURN1

**Technology File**

prTurnViaRules(
  "TURN1" M1 ("vertical" "horizontal") )
  "TURN2" M2 ("vertical" "horizontal") )
  "TURN3" M3 ("vertical" "horizontal")
)

---

**NONDEFAULTRULE Section**

The NONDEFAULTRULE section in the LEF file define widths for regular wiring wider than the default regular wiring.

NONDEFAULTRULE ruleName1
  {LAYER routinglayerName
    WIDTH width
    SPACING minSpacing
    [WIREEXTENSION value ;]
    [RESISTANCE RPERSQ value;]
    [CAPACITANCE CPERSQDIST value;]
    [EDGECAPACITANCE value;]
    END routinglayerName}...

VIA viaName
  [FOREIGN foreignCellName
    [pt [orient]];]
  [RESISTANCE value;]
  [PROPERTY(propName propVal) ;]..
Non-default layer rules after mapping to the technology file are written in the prNonDefaultRules statement in the prRules section.

Example

- **LEF File**

  NONDEFAULTRULE RULE1
  LAYER M1
  SPACING 2.2 ;
  WIDTH 0.5 ;
  RESISTANCE RPERSQ 2.2 ;
  CAPACITANCE CPERSQDIST 3.3 ;
  EDGECAPACITANCE 4.5 ;
END M1
...
END RULE1

- **Technology File**

  prNonDefaultRules(
    ;(ruleName ((layerName width spacing notch wireExt cap resistance edgeCap) ...)
    ;( viaNameList )
    ;( (viaLayer1 viaLayer2 spacing t/nil) ... ))
  ( "RULE1" 
    ( "M1" 0.500 2.200 _NA_ _NA_ 3.300000 2.200000 4.500000)
    ( "ndVIA12" )
  ) ; end of prNonDefaultRules.

Non-default vias, after being mapped to the technology file, are written in the ruleContactDevice subsection under the devices section in the technology file. For details, refer to ruleContactDevice in the via section.

The non-default via properties, after being mapped to the technology file, are written as tfcDefineDeviceProp under the devices section.
Example

- **LEF File**

  NONDEFAULTRULE RULE1
  ......
  VIA ndVIA12
  PROPERTY count 50 ;
  LAYER M2 ;
  RECT -2.2 -2.2 2.2 2.2 ;
  LAYER CUT12 ;
  RECT -0.8 -0.8 0.8 0.8 ;
  LAYER M1 ;
  RECT -2.0 -2.0 2.0 2.0 ;
  END ndVIA12
  ......
  END RULE1

- **Technology File**

  tfcDefineDeviceProp(
    ;...
    (symbolic "ndVIA12" "count" 50)
    ;...
  ); end of tfcDefineDeviceProp

The dfII technology file specifies the **SAMENET** nondefault rules in the nondefault rules section of the **Place and Route Rules** class as shown below:

- Rule name
- Layer name
- Via name
- Layer width
- SAMENET minSpace for same layer
- Layer spacing
- SAMENET minSpace for different layers
- Can vias on these layers be stacked? (t or nil)
**SPACING Section**

The SPACING section in the LEF file provides for the same net spacing rules and stacked vias.

```
SPACING
  {SAMENET
    layerName layerName minSpace
    [STACK] ;}...
END SPACING
```

In the dfII technology file, the spacing rules section of the Physical Rules class specifies same net spacing.

```
    spacingRules(
        ( sameNet metall metall 0.600000 )
        ...,
        ( sameNet via via2 0.600000 )
        ...
    )
```

The prStackVias section of the Place and Route Rules class lists stackable via layers.

```
prStackVias(
    (via via2 .6
    )
    ...
)
```

**SITE Section**

The SITE section defines the types and properties of placement sites available in the library for defining rows.

```
{ SITE siteName
    CLASS { CORE | PAD }
    SYMMETRY { X | Y | R90 }...
    SIZE width BY height ;
    END siteName }
```

Each SITE section in a LEF file maps to a cell in the dfII library with the prCellType site. Each site cell has the following four cellviews:

- abstract
The abstract cellview defines and represents the site.

- **cannnotoccupy**

  dfII Gate Ensemble places `cannnotoccupy` cellviews over site abstracts to show graphically the sites that cannot be occupied.

- **canplace**

  dfII Gate Ensemble places `canplace` cellviews over site abstracts to show graphically the sites that have are valid origins.

- **layout**

  The dfII database does not use the layout cellviews for sites.

Exporting a LEF file generates a `SIZE` statement based on the site bounding box on the `prBoundary` boundary layer-purpose pair. *File – Export – Lef* looks for a single rectangular geometry on this layer. If you create additional geometries, *File – Export – LEF* ignores them.

<table>
<thead>
<tr>
<th>LEF</th>
<th>dfII</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SITE siteName</strong></td>
<td>Cell name of site cellviews</td>
</tr>
<tr>
<td><strong>CLASS</strong></td>
<td><code>prCellClass</code> of site cellview</td>
</tr>
<tr>
<td><strong>CORE</strong></td>
<td><code>core</code></td>
</tr>
<tr>
<td><strong>PAD</strong></td>
<td><code>pad</code></td>
</tr>
<tr>
<td><strong>SYMMETRY</strong></td>
<td><code>symmetry</code> of site cellview</td>
</tr>
<tr>
<td>X</td>
<td><code>x</code></td>
</tr>
<tr>
<td>Y</td>
<td><code>y</code></td>
</tr>
<tr>
<td>R90</td>
<td><code>r90</code></td>
</tr>
<tr>
<td>X Y</td>
<td><code>x y</code></td>
</tr>
<tr>
<td>X R90</td>
<td><code>x r90</code></td>
</tr>
<tr>
<td>Y R90</td>
<td><code>y r90</code></td>
</tr>
<tr>
<td>X Y R90</td>
<td><code>x y r90</code></td>
</tr>
<tr>
<td><strong>SIZE width BY height</strong></td>
<td>Width and height of the site bounding box</td>
</tr>
</tbody>
</table>

**ARRAYS Section**

**Note:** The `ARRAYS` section is applicable only for Gate Ensemble.
The **ARRAY** section defines the library base array. A base array includes arrays of sites and site patterns that define the core and the I/O areas of the chip block. It also includes a grid of routing tracks across the design.

```
{ ARRAY arrayname
    {SITE sitePattern;
     CANPLACE sitePattern;
     CANNOTOCCUPY sitePattern;
     TRACKS trackPattern;
     FLOORPLAN floorplanName
        {{CANPLACE | CANNOTOCCUPY}
             sitePattern; }...
        END floorplanName}....
    [GCELLGRID gcellPattern;]...
    [DEFAULTCAP tableSize
        {MINPINS numPins WIRECAP cap ; }...
        END DEFAULTCAP]
    END arrayName }....
```

Each **ARRAY** section in the LEF file maps to a base array cellview in the dfII library. The dfII database stores the **FLOORPLAN** statements as separate base array cellviews with the pattern of **canplace** and **cannotoccupy** cellviews specified in the **FLOORPLAN** statement. The sites and grids of the floorplan base array are the same as those of the parent base array.

<table>
<thead>
<tr>
<th>LEF</th>
<th>dfII</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRAY arrayName</td>
<td>Cell name of base array cellview.</td>
</tr>
<tr>
<td>SITE sitePattern</td>
<td>Site layout cellviews placed in the base array cellview</td>
</tr>
<tr>
<td>CANPLACE sitePattern</td>
<td>Site <strong>canplace</strong> cellviews placed in the base array cellview</td>
</tr>
<tr>
<td>CANNOTOCCUPY sitePattern</td>
<td>Site <strong>cannotoccupy</strong> cellviews placed in the base array cellview</td>
</tr>
<tr>
<td>TRACKS trackPattern</td>
<td>Routing grid of the base array cellview</td>
</tr>
<tr>
<td>FLOORPLAN floorplanName</td>
<td>Additional base array with specified floorplan</td>
</tr>
<tr>
<td>GCELLGRID gcellPattern</td>
<td>The <strong>gCell</strong> grid of the base array cellview</td>
</tr>
</tbody>
</table>
MACRO Section

The MACRO section in the LEF file contains definitions for the macrocell models in the library. Each macro definition maps to an abstract cellview in the dfII library.

```
{ MACRO macroName
  macrocell model data
  pin data...
  obstruction data...
  END macroName }...
```

LEF timing data is not imported into dfII. The dfII database uses timing data in the CTLF format. If you have timing views, you can translate them to CTLF.

<table>
<thead>
<tr>
<th>LEF</th>
<th>dfII</th>
</tr>
</thead>
<tbody>
<tr>
<td>MACRO macroName</td>
<td>Cell name of the cellview</td>
</tr>
<tr>
<td>View name of cellview is abstract</td>
<td></td>
</tr>
</tbody>
</table>

Macro Cell Models

```
[ CLASS { COVER [BUMP] | RING | BLOCK [BLACKBOX] | PAD
  [INPUT | OUTPUT | INOUT | POWER | SPACER | AREAIO]
  | CORE
  [FEEDTHRU | TIEHIGH | TIELOW | SPACER | ANTENNACELL]
  | ENDCAP
  {PRE | POST | TOPLEFT | TOPRIGHT | BOTTOMLEFT
  | BOTTOMRIGHT}
  } ; ]
[ SOURCE { USER | GENERATE | BLOCK } ; ]
[ FOREIGN
  foreignCellName [pt [orient]] ; ]...
[ ORIGIN pt ; ]
[ EEQ macroName ; ] [ LEQ macroName ; ]
SIZE width BY height ;
[ SYMMETRY ( X | Y | R90 ) ... ; ]
[ POWER powerConsumption ; ]
[ SITE siteName ; ]
{PIN statement}...
[OBS statement]...
[TIMING statement]...
[PROPERTY {propName propVal} ... ; ]...
```

**Note:** For Gate Ensemble, the Site statement in Macro Cell Models is written as
( SITE siteName origX origY orient stepPattern)...

<table>
<thead>
<tr>
<th>LEF</th>
<th>dfil</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASS COVER, RING, or BLOCK</td>
<td>prCellType of the macro cellview is macro</td>
</tr>
<tr>
<td>COVER</td>
<td>prCellClass is cover</td>
</tr>
<tr>
<td>RING</td>
<td>prCellClass is ring</td>
</tr>
<tr>
<td>BLOCK</td>
<td>prCellClass is block</td>
</tr>
<tr>
<td>BUMP</td>
<td>prCellClass is coverBump</td>
</tr>
<tr>
<td>BLACKBOX</td>
<td>prCellClass is blackBox</td>
</tr>
<tr>
<td>CLASS PAD</td>
<td>prCellType of the macro cellview is IO</td>
</tr>
<tr>
<td>INPUT</td>
<td>prCellClass is nil</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>prCellClass is input</td>
</tr>
<tr>
<td>INOUT</td>
<td>prCellClass is output</td>
</tr>
<tr>
<td>POWER</td>
<td>prCellClass is inOut</td>
</tr>
<tr>
<td>SPACER</td>
<td>prCellClass is power</td>
</tr>
<tr>
<td>AREAIO</td>
<td>prCellClass is spacer</td>
</tr>
<tr>
<td></td>
<td>prCellClass is areaIO</td>
</tr>
<tr>
<td>CLASS CORE</td>
<td>prCellType of macro cellview is standard</td>
</tr>
<tr>
<td>FEEDTHRU</td>
<td>prCellClass is nil</td>
</tr>
<tr>
<td>TIEHIGH</td>
<td>prCellClass is jumper</td>
</tr>
<tr>
<td>TILOW</td>
<td>prCellClass is tieHigh</td>
</tr>
<tr>
<td>SPACER</td>
<td>prCellClass is tiLow</td>
</tr>
<tr>
<td>ANTENNACELL</td>
<td>prCellClass is spacer</td>
</tr>
<tr>
<td></td>
<td>prCellClass is antennaCell</td>
</tr>
<tr>
<td>CLASS ENDCAP</td>
<td>prCellType of macro cellview is standard for PRE and POST. For TOLEFT, TOPRIGHT, BOTTOMLEFT, BOTTOMRIGHT, and prCellType is corner.</td>
</tr>
<tr>
<td>PRE</td>
<td>prCellClass is postEndcap</td>
</tr>
<tr>
<td>POST</td>
<td>prCellClass is topLeftEndcap</td>
</tr>
<tr>
<td>TOPLEFT</td>
<td>prCellClass is topRightEndcap</td>
</tr>
<tr>
<td>TOPRIGHT</td>
<td>prCellClass is bottomLeftEndcap</td>
</tr>
<tr>
<td>BOTTOMLEFT</td>
<td>prCellClass is bottomRightEndcap</td>
</tr>
</tbody>
</table>

ORIGIN pt

For cells whose origin is not at 0,0, File – Export – LEF uses these coordinates to offset all cell geometries such that the origin of the cell falls at the lower left corner of the bounding rectangle. File – Import reverses this process.
### Design Data Translator's Reference

**Translating LEF and DEF Files**

<table>
<thead>
<tr>
<th>LEF</th>
<th>dfII</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOREIGN foreignCellName</td>
<td>gec3Foreign property of the macro cellview. Foreign cell name, stored as string.</td>
</tr>
<tr>
<td>EEQ name</td>
<td>eeq of the macro cellview</td>
</tr>
<tr>
<td>LEQ name</td>
<td>leq of the macro cellview</td>
</tr>
<tr>
<td>SIZE width BY height</td>
<td>Width and height of the bounding box on the prBoundary, and boundary layer-purpose pair of the macro cellview.</td>
</tr>
<tr>
<td>SYMMETRY</td>
<td>symmetry and allowedOrient of macro cellview.</td>
</tr>
<tr>
<td>X</td>
<td>xr0, mx</td>
</tr>
<tr>
<td>Y</td>
<td>yr0, my</td>
</tr>
<tr>
<td>R90</td>
<td>r90r0, r90, r180, r270</td>
</tr>
<tr>
<td>X Y</td>
<td>xyr0, my, r180, mx</td>
</tr>
<tr>
<td>X R90</td>
<td>xyr90r0, r90, r180, r270, mx, myr90, my, mr90</td>
</tr>
<tr>
<td>Y R90</td>
<td>mxr90</td>
</tr>
<tr>
<td>X Y R90</td>
<td>yr90r0, r90, r180, r270, mx, myr90, my, mr90</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>POWER powerCons</td>
<td>powerConsumption of macro cellview</td>
</tr>
<tr>
<td>SITE siteName</td>
<td>Name of the site cellview that determines legal placement locations for the macro.</td>
</tr>
</tbody>
</table>

**For Gate Ensemble:**

For each SITE statement in the macro definition, *File – Import – LEF* creates an array of sites in the abstract view of the cell. *File – Export – LEF* creates a SITE statement for each site array and site instance placed in the macro cellview.

**PROPERTY propName propVal**

### Note:
■ *File – Import – LEF* and *File – Export – LEF* do not support the *SOURCE* statement.

**Creating the Rectilinear Boundary of a Macro**

During LEF In, the boundary of a macro is usually created from the *SIZE* statement of the *MACRO*. This boundary is always rectangular even if the actual shape of the macro is rectilinear. A new SKILL variable *'createBoundaryAsOverlap'* has been provided to create the rectilinear boundary of a *MACRO* during LEF In.

To create a rectilinear boundary of a macro during LEF In, set the SKILL variable *'createBoundaryAsOverlap'* to *t*. In addition, specify a placement obstruction in the *OBS* statement of the *MACRO* in the LEF file. On setting the value of SKILL variable *'createBoundaryAsOverlap'* to *t*, the rectilinear boundary of a macro gets its coordinates from the *POLYGON* statement, which is under the *OBS* statement of *MACRO*.

If you specify the value of the SKILL variable *'createBoundaryAsOverlap'* as *nil*, then a rectangular boundary of a macro is created.

**Example**

```plaintext
MACRO ABC
  SIZE 20 BY 20
  ....
  ....
  OBS
  POLYGON 0 0 10 10 10 20 10 20 0 0 20
END
END ABC
```

In the above example, if you set the SKILL variable *'createBoundaryAsOverlap'* to *nil*, then boundary of macro would be rectangular of size equal to 20 BY 20. If the SKILL variable *'createBoundaryAsOverlap'* is equal to *t* then boundary of macro would be rectilinear with coordinates coming from the *POLYGON* statement within the *OBS* statement.
PIN Section

The PIN section in LEF defines a pin for a macro.

PIN pinName
[ TAPERRULE ruleName; ]
[ FOREIGN foreignPinName; ]
[ FOREIGN foreignPinName
    STRUCTURE [pt [orientation]] ; ]...
[ LEQ pinName; ]
[ DIRECTION
    { INPUT | OUTPUT [TRISTATE] | INOUT | FEEDTHRU}; ]
[ USE { SIGNAL | ANALOG | POWER | GROUND | CLOCK }; ]
[ SHAPE {ABUTMENT | RING | FEEDTHRU}; ]
[ MUSTJOIN pinName; ]
[ OUTPUTNOISEMARGIN statement; ]
[ OUTPUTRESISTANCE statement; ]
[ INPUTNOISEMARGIN statement; ]
[ POWER powerConsumption; ]
[ LEAKAGE current; ]
[ CAPACITANCE pinCapacitance; ]
[ RESISTANCE pinResistance; ]
[ MAXLOAD capvalue; ]
[ PULLDOWNRES resistance; ]
[ TIEOFFR res; ]
[ VHI voltage; ]
[ VLO voltage; ]
[ RISETHRESH capacitance; ]
[ FALLTHRESH capacitance; ]
[ RISESATCUR current; ]
[ FALLSATCUR current; ]
[ RISEVOLTAGETHRESHOLD voltage; ]
[ FALLVOLTAGETHRESHOLD voltage; ]
[ MAXDELAY delayTime; ]
[ CURRENTSOURCE {ACTIVE | RESISTIVE}; ]
[ IV_TABLES lowTableName hiTableName ]
[ ANTENNASIZE value; ]
{ PORT [CLASS CORE] layerGeometries... END }...
[PROPERTY { propName propVal}...; ]...
[ANTENNAPARTIALMETALAREA value [LAYER layerName]; ]...
[ANTENNAPARTIALMETALSIDERAAREA value [LAYER layerName]; ]...
[ANTENNAPARTIALCUTAREA value [LAYER layerName]; ]...
[ANTENNADIFFAREA value [LAYER layerName]; ]...
[ANTENNAOxIDE{1,2,3,4}; ]
[ANTENNAGATEAREA value [LAYER layerName]; ]...
[ANTENNAMAXAREACAR value LAYER layerName; ]...
[ANTENNAMAXSIDEAREACAR value LAYER layerName ]; ]...
[ANTENNAMAXCUTCAR value LAYER layerName ]; ]...
END pinName

Supported upto LEF version 5.3

Supported upto LEF version 5.3
The dfII database uses timing data in the CTLF format. If you have timing views from the 4.3 release of dfII, you can translate them to CTLF.

Pins are called terminals in dfII. Physical geometries (ports) are called pins. For each PIN statement, except for mustjoin pins, File – Import – LEF creates a terminal on the macro cell view with the specified pins. The dfII database interprets mustjoin pins as pins (ports) on the same terminal.

<table>
<thead>
<tr>
<th>LEF</th>
<th>dfII</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIN pinName</td>
<td>Terminal name</td>
</tr>
<tr>
<td>FOREIGN foreignPinName</td>
<td>gec3Foreign property of the terminal, stored as string</td>
</tr>
<tr>
<td>STRUCTURE pt orientation</td>
<td>leg property of terminal, stored as a string</td>
</tr>
<tr>
<td>LEQ pinName</td>
<td>direction of terminal</td>
</tr>
<tr>
<td>DIRECTION</td>
<td></td>
</tr>
<tr>
<td>INPUT</td>
<td>input</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>output</td>
</tr>
<tr>
<td>TRISTATE</td>
<td>not supported</td>
</tr>
<tr>
<td>INOUT</td>
<td>input/output</td>
</tr>
<tr>
<td>FEEDTHRU</td>
<td>jumper</td>
</tr>
<tr>
<td>USE</td>
<td>use of terminal</td>
</tr>
<tr>
<td>SIGNAL</td>
<td>signal</td>
</tr>
<tr>
<td>ANALOG</td>
<td>analog</td>
</tr>
<tr>
<td>POWER</td>
<td>power</td>
</tr>
<tr>
<td>GROUND</td>
<td>ground</td>
</tr>
<tr>
<td>CLOCK</td>
<td>clock</td>
</tr>
</tbody>
</table>
### Design Data Translator's Reference
#### Translating LEF and DEF Files

<table>
<thead>
<tr>
<th>LEF</th>
<th>dfII</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHAPE</td>
<td>terminal shape</td>
</tr>
<tr>
<td>ABUTMENT</td>
<td>abutment</td>
</tr>
<tr>
<td>RING</td>
<td>ring</td>
</tr>
<tr>
<td>FEEDTHRU</td>
<td>feedthru</td>
</tr>
<tr>
<td>MUSTJOIN pinName</td>
<td>Causes a LEF pin to be interpreted as a dfII must-connect pin on terminal pinName</td>
</tr>
<tr>
<td>POWER powerCons</td>
<td>powerConsumption of terminal</td>
</tr>
<tr>
<td>CAPACITANCE pinCap</td>
<td>capacitance of terminal</td>
</tr>
<tr>
<td>MAXLOAD capvalue</td>
<td>maxload of terminal</td>
</tr>
<tr>
<td>RESISTANCE pinResistance</td>
<td>resistance of the terminal</td>
</tr>
<tr>
<td>ANTENNASIZE value</td>
<td>pinAntenna property of terminal</td>
</tr>
<tr>
<td>PORT layerGeometries</td>
<td>Pin geometries in macro abstract cellview</td>
</tr>
<tr>
<td>PROPERTY propName propVal</td>
<td>Property of a terminal</td>
</tr>
<tr>
<td>ANTENNAMODEL OXIDE</td>
<td>OXIDE1 and OXIDE2 are supported. OXIDE3 and OXIDE4 are ignored.</td>
</tr>
<tr>
<td>ANTIENNA PARTIAL METAL AREA value [LAYER layerName]</td>
<td>Mapped as a hierarchical property. The parent property name is ANTIENNA PARTIAL METAL AREA. The hierarchical property name is layerName or DEFAULT (if layerName is not specified), the hierarchical property type is float, and the hierarchical property value is value.</td>
</tr>
<tr>
<td>ANTIENNA PARTIAL METAL SIDE AREA value [LAYER layerName]</td>
<td>Mapped as a hierarchical property. The parent property name is ANTIENNA PARTIAL METAL SIDE AREA. The hierarchical property name is layerName or DEFAULT (if layerName is not specified), the hierarchical property type is float, and the hierarchical property value is value.</td>
</tr>
<tr>
<td>ANTIENNA GATE AREA value [LAYER layerName]</td>
<td>Mapped as a hierarchical property. The parent property name is ANTIENNA GATE AREA. The hierarchical property name is layerName or DEFAULT (if layerName is not specified), the hierarchical property type is float, and the hierarchical property value is value.</td>
</tr>
<tr>
<td>LEF</td>
<td>dfII</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td><strong>ANTENNADIFFAREA</strong> value [LAYER layerName]</td>
<td>Mapped as a hierarchical property. The parent property name is <strong>ANTENNADIFFAREA</strong>. The hierarchical property name is <strong>layerName</strong> or DEFAULT (if <strong>layerName</strong> is not specified), the hierarchical property type is <strong>float</strong>, and the hierarchical property value is <strong>value</strong>.</td>
</tr>
<tr>
<td><strong>ANTENNAMAXAREACAR</strong> value LAYER layerName</td>
<td>Mapped as a hierarchical property. The parent property name is <strong>ANTENNAMAXAREACAR</strong>. The hierarchical property name is <strong>layerName</strong> or DEFAULT (if <strong>layerName</strong> is not specified), the hierarchical property type is <strong>float</strong>, and the hierarchical property value is <strong>value</strong>.</td>
</tr>
<tr>
<td><strong>ANTENNAMAXSIDEAREACAR</strong> value LAYER layerName</td>
<td>Mapped as a hierarchical property. The parent property name is <strong>ANTENNAMAXSIDEAREACAR</strong>. The hierarchical property name is <strong>layerName</strong> or DEFAULT (if <strong>layerName</strong> is not specified), the hierarchical property type is <strong>float</strong>, and the hierarchical property value is <strong>value</strong>.</td>
</tr>
<tr>
<td><strong>ANTENNAPARTIALCUTAREA</strong> value [LAYER layerName]</td>
<td>Mapped as a hierarchical property. The parent property name is <strong>ANTENNAPARTIALCUTAREA</strong>. The hierarchical property name is <strong>layerName</strong> or DEFAULT (if <strong>layerName</strong> is not specified), the hierarchical property type is <strong>float</strong>, and the hierarchical property value is <strong>value</strong>.</td>
</tr>
<tr>
<td><strong>ANTENNAMAXCUTCAR</strong> value LAYER layerName</td>
<td>Mapped as a hierarchical property. The parent property name is <strong>ANTENNAMAXCUTCAR</strong>. The hierarchical property name is <strong>layerName</strong> or DEFAULT (if <strong>layerName</strong> is not specified), the hierarchical property type is <strong>float</strong>, and the hierarchical property value is <strong>value</strong>.</td>
</tr>
</tbody>
</table>

**Layer name**

**PATH**

**RECT**

**POLYGON**

dfII creates obstructions with the specified geometry on the (**layerName**, boundary) layer-purpose pair in the macro abstract cellview.

dfII also creates obstructions on the (**layerName**, drawing) layer-purpose pair in the macro layout cellview.
Note: OUTPUTNOISEMARGIN, INPUTNOISEMARGIN, LEAKAGE, PULLDOWNRES, TIEOFFRES, VHI, VLO, RISETHRESH, FALLTHRESH, RISEATCURR, RISEVOLTAGETHRESHOLD, FALLVOLTAGETHRESHOLD, MAXDELAY, and IV_TABLES are stored as a list of properties with the property name lefobjProplist.

dfII Pin Model

Macrocell pins map to the models shown below.

- The dfII database interprets geometries defined within the same port in LEF as strong-connect pins. Strong-connect pins are groups of geometries that a router can include as part of the connectivity of a net. The router can assume that any routing connected to one strong-connect pin is also connected to the other.

```
Routing for net IN

PIN IN
PORT LAYER metal1 ;
RECT 0 7.5 7.5 9.5 ;
RECT 0 5 2.5 7.5 ;
END END IN
```

```
Geometries in the same port
```
The dfII database uses subnets to model the connection between strong-connect pins, as shown below:

Terminal: *IN*

Net: *IN*  
File – Export uses the subnet name as PIN *pinName*.

Subnet: *IN*  
Subnet of subnet  
 Pin P  
 Pin Q  
Strong-connect pins

The dfII database interprets geometries defined within different ports in LEF as weak-connect pins. Weak-connect pins are groups of geometries that a router cannot assume to be connected but which let the router connect to any one geometry in the group. For example, if geometries P and Q are weakly connected, the router can route to P or Q but not both.

PIN IN  
PORT LAYER metal1;  
  RECT 0 7.5 7.5 9.5;  
END  
PORT LAYER metal1;  
  RECT 0 0 5 2;  
END END IN

The dfII database uses subnets to model the connection between weak-connect pins, as shown:

Terminal: *IN*

Net: *IN*  
File – Export uses subnet name as PIN *pinName*.

Subnet: *IN*  
Subnet of subnet  
 Pin P  
 Subnet of subnet  
 Pin Q  
Weak-connect pins

Geometries in different ports
The dfll database interprets mustjoin pins in the LEF file as pins (ports) of the same terminal. The first pin defined in the LEF file is mapped to a terminal. Subsequent pins mustjoined to the first pin, including pins in a mustjoin ring, map to the same terminal. The dfll database mustjoin or mustconnect pins are groups of geometries that a router must connect together because the geometries do not connect within the cell. For example, if geometries P and Q are mustconnect pins, the router must connect P and Q together.

The dfll database uses subnets to model must-connect pins as shown:

**Terminal:** IN

**Net:** IN

**Subnet:** IN

- Subnet of subnet
  - Pin P
  - Must-connect pins
  - Subnet IN.1
  - Subnet of subnet
  - Pin Q
- File – Export uses subnet names as PIN pinName.
LEF to dfII Pin Mapping Example

```
PIN IN
PORT LAYER metal1 ;
    RECT 0 7.5 7.5 9.5 ; # rect A
    RECT 0 5 2.5 7.5 ; # rect B
END
PORT LAYER metal1 ;
    RECT 10 5 12.5 9.5 ; # rect D
END
PORT LAYER metal1 ;
    RECT 7.5 0 12.5 2 ; # rect E
END
PIN IN.1 MUSTJOIN IN ;
PORT LAYER metal1 ;
    RECT 0 0 5 2 ;       # rect C
END
```

**Must connects**
- subnet name = IN
- subnet name = IN

**Weak connects**
- subnet
- subnet
- subnet

**Strong connect**
- Rectangle A
- Rectangle B
- Rectangle C
- Rectangle D
- Rectangle E
OBS Section

The OBS section in the LEF file defines sets of obstructions on a macro. Obstructions are also called blockages.

```lef
OBS
  { LAYER layerName [SPACING minSpacing | DESIGNRULEWIDTH widthValue];
   { [ WIDTH width ; ]
   { PATH pt... ;
   | PATH ITERATE pt [ pt ] iteratePattern
   | RECT pt pt ;
   | RECT ITERATE pt pt iteratePattern
   | POLYGON pt pt pt pt... ;
   | POLYGON ITERATE polyCoord iteratePattern
   | VIA pt viaName ;
   | VIA ITERATE pt viaName iteratePattern
   } }... }...
END
```


<table>
<thead>
<tr>
<th>LEF</th>
<th>dfII</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAYER layerName</td>
<td>Layer name</td>
</tr>
<tr>
<td>PATH</td>
<td>The dfII database creates obstructions with the specified geometry on the (layerName, boundary) layer-purpose pair in the macro abstract cellview.</td>
</tr>
<tr>
<td>RECT</td>
<td>The dfII database also creates obstructions on the (layerName, drawing) layer-purpose pair in the macro layout cellview.</td>
</tr>
<tr>
<td>POLYGON</td>
<td></td>
</tr>
<tr>
<td>VIA pt viaName</td>
<td>Via instance</td>
</tr>
<tr>
<td>DESIGNRULEWIDTH</td>
<td>Mapped as the DESIGNRULEWIDTH hierarchical property in the PropertyBag of the cellview corresponding to the macro.</td>
</tr>
<tr>
<td>widthValue</td>
<td></td>
</tr>
<tr>
<td>SPACING minSpacing</td>
<td>Mapped as the MINSPACING property in the PropertyBag of the cellview corresponding to the macro.</td>
</tr>
</tbody>
</table>
ANTENNA Statements

The INPUTPINANTENNASIZE, OUTPUTPINANTENNASIZE, and INOUTPINANTENNASIZE statements in LEF set default upper limits used by the VERIFY ANTENNA command in Silicon Ensemble.

[[INPUTPINANTENNASIZE | INOUTPINANTENNASIZE | OUTPUTPINANTENNASIZE] value ;].
[ANTENNAINPUTGATEAREA value;]
[ANTENNAINOUTDIFFAREA value;]
[ANTENNAOUTPUTDIFFAREA value;] Supported in LEF 5.4

These statements map to the library property pinAntenna. The value of the pinAntenna property is a list of three floating point numbers, corresponding to the values of INPUTPINANTENNASIZE, OUTPUTPINANTENNASIZE, and INOUTPINANTENNASIZE, in the specified order. In addition, they are mapped as a property of name equal to objProperty on the database object PropertyBag of the library. The property type is a list and it consists of various properties as the elements of that list. The three elements INPUTPINANTENNASIZE, OUTPUTPINANTENNASIZE, and INOUTPINANTENNASIZE map to the technology file as (“INPUTPINANTENNASIZE”, value), (“OUTPUTPINANTENNASIZE”, value), and (“INOUTPINANTENNASIZE”, value).

If any of the antenna size statements is absent from the LEF file, the dfll library property will use a value of -1.0. The three antenna sizes ANTENNAINPUTGATEAREA, ANTENNAINOUTDIFFAREA, and ANTENNAOUTPUTDIFFAREA map to the technology file statement as follows:

controls(
    techParams(
        ; (parameter value)
        (LEFDEF_INPUTPINANTENNASIZE value)
        (LEFDEF_INOUTPINANTENNASIZE value)
        (LEFDEF_OUTPUTPINANTENNASIZE value)
    ); techParams
)

The default values for these parameters would be 1.0, -1.0, and -1.0 respectively.

Note: The three antenna sizes ANTENNAINPUTGATEAREA, ANTENNAINOUTDIFFAREA, and ANTENNAOUTPUTDIFFAREA will not be written to the technology file if the LEF version is greater than or equal to 5.5.

MANUFACTURINGGRID Statement

Manufacturing grid is used to control the distance between the two snap grids.
The snap grid controls pointer movement in the interactive drawing and editing modes. When the snap grid is defined, the pointer snaps to the closest grid point as you move it within the work area.

Distance between the two snap grids must be a multiple of the manufacturing grid. The value specified for the manufacturing grid should be positive. The numeric range is float.

\[ \text{MANUFACTURINGGRID } \text{value}; \]

After mapping to the technology file, the manufacturing grid is written in the `techParams` section, which is a subsection of the `controls` section of the technology file. In addition, it is also written in the `mfgGridResolution` section, which is a subsection of the `physicalRules` section of the technology file. Their syntax is shown below.

```plaintext
controls(
  techParams(
    ;(parameter value)
    (LEFDEF_MANUFACTURINGGRID value)
  );
);

physicalRules(
  ...
  mfgGridResolution(
    (value)
  );mfgGridResolution
);
```

During LEF Out, if the manufacturing grid values are different in the `techParams` and `mfgGridResolution` sections, the value specified in the `mfgGridResolution` section takes precedence and is written in the LEF file. Following warning message is displayed:

*WARNING* Value of Manufacturing Grid at controls section differs from physicalRules section: Taking the value from physicalRules section.

**USEMINSPACING Statement**

Defines how minimum spacing is calculated for an obstruction (blockage) or pin geometries. It is mapped as the `USEMINSPACINGOBS` or `USEMINSPACINGPIN` property on the database object `PropertyBag` in the library.

```plaintext
USEMINSPACING {OBS | PIN} {ON | OFF};...
```

In the above syntax, \( \text{OBS|PIN} \) specifies whether the minimum spacing is applied to pin geometries or obstructions. The option \( \text{ON|OFF} \) specifies the calculation method for minimum
spacing. If you specify ON, the spacing is the minimum layer spacing. If you specify OFF, the spacing is in the range that corresponds to the minimum width of the pin or the obstruction.

**CLEARANCEMEASURE Statement**

Specifies the clearance distance used for spacing rule checks. It is mapped as the CLEARANCEMEASURE property on the database object PropertyBag of the library.

```
CLEARANCEMEASURE {MAXXY | EUCLIDEAN};
```

In the above syntax, MAXXY uses the larger X and Y distances for spacing rule checks. Similarly, EUCLIDEAN uses Euclidean distance for spacing rule check. The default value is EUCLIDEAN.

**DEF Data Map**

This topic describes the following DEF sections and shows how they map to dfII:

- VERSION Section on page 388
- PROPERTYDEFINITIONS Section on page 390
- VIAS Section on page 390
- REGIONS Section on page 391
- COMPONENTS Section on page 392
- PINS Section on page 394
- IOTimings Section on page 399
- NETS Section on page 401
- SPECIALNETS Section on page 406
- GROUPS Section on page 412
- CONSTRAINTS Section on page 413
- BLOCKAGES Section on page 414
- SLOTS Section on page 416
- FILLS Section on page 417
- SCANCHAINS Section on page 418
**Note:** *File – Import – DEF* and *File – Export – DEF* do not support the `BEGINEXT` statement.

### VERSION Section

The following statements in DEF contain global information about the design.

```
[ VERSION version ; ]
[ NAMESCASESENSITIVE (OFF | ON) ; ]
[ DIVIDERCHAR "character" ; ]
[ BUSBITCHARS "delimiterPair" ; ]
[ ARRAY arrayName ]
[ DESIGN designName ; ]
[ TECHNOLOGY techName ; ]
[ HISTORY anyText ; ]...
[ UNITS DISTANCE MICRONS convertFactor ; ]
[ FLOORPLAN {DEFAULT | floorplanName} ; ]
[ ROW rowType origX origY orient stepPattern ; ]...
[ TRACKS
  {X | Y} start stepPattern
  LAYER layerName ; ]
[ GCELLGRID
  X start stepPattern
  Y start stepPattern ; ]
[ DIEAREA ( x y x y ) ;]
[ DEFAULTCAP tableSize
  {MINPINS numPins WIRECAP cap ; ]
```

---

**DEF**

<table>
<thead>
<tr>
<th>DEF</th>
<th>dfll</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIVIDERCHAR “character”</td>
<td><em>File – Export – DEF</em> always writes DIVIDERCHAR as “</td>
</tr>
<tr>
<td>BUSBITCHARS “delimiterPair”</td>
<td>The dfll tools use delimiterPair to identify bus bit characters for name mapping. <em>File – Import – DEF</em> writes BUSBITCHARS as “()” for nets if the environmental variable defCONNMAP is not set to “VXLDEF”. If defCONNMAP is set to “VXLDEF”, <em>File – Export – DEF</em> replaces BUSBITCHARS by “&lt;&gt;”.</td>
</tr>
</tbody>
</table>
### Design Data Translator's Reference

**Translating LEF and DEF Files**

#### File – Import – DEF
- **ARRAY arrayName**
  - Cell name of the base array instantiated in the design cellview.
- **DESIGN designName**
  - Cell name of a design cellview.
- **TECHNOLOGY techName**
  - File – Import – DEF skips the technology statement.
  - File – Export – DEF writes the technology statement as the current library name in the output DEF file.
- **HISTORY text**
  - File – Import – DEF ignores the HISTORY statement.
- **UNITS DISTANCE MICRON DEFconvertFactor;**
  - File – Import – DEF checks the DEFconvertFactor with the library units. If these are not equal, a warning is displayed and the value of DEFconvertFactor is set to its default value, that is, 100.
  - File – Export – DEF searches for the gec3DBUPerUU property in the technology library. Possible values of the DEFconvertFactor can be 100, 200, 1000, and 2000. In case of any other values, a warning is displayed and the default value is used for further calculations.
- **FLOORPLAN DEFAULT floorplanName**
  - Mapped as a property on a cellview with the property name floorPlanName, property type dbcStringType, and property value as the name specified.
  - File – Export – DEF can generate a DEF floorPlanName from the floorPlanName property of the design cellview.
- **ROW rowType originX originY orient stepPattern**
  - Row of sites in the design cellview.
- **TRACKS**
  - Routing grid of the design cellview.
- **GCELLGRID X start stepPattern Y start stepPattern**
  - gCell grid of the design cellview.
- **DIEAREA (x y x y)**
  - Corner points of the bounding box in the design cellview.
**PROPERTYDEFINITIONS** Section

The **PROPERTYDEFINITIONS** section of the DEF file lists the properties you can assign to various design objects.

```plaintext
PROPERTYDEFINITIONS
  objectType propName propType [RANGE # #] [value | stringValue]; ...
END PROPERTYDEFINITIONS
```

*File – Export – DEF* generates the following **HISTORY** statement from the cell history data:

```plaintext
HISTORY
design data:
Creator: name
data path: full path
Library: name
Cell Name: name
View Name: name
Version: name
Date: created date
;
```

**VIAS Section**

The **VIAS** section in the DEF file contains the name and geometries of each via model used in regular and special wiring. Some of these vias are defined in the LEF file, but many are generated from via rules as they are needed by routers.
### VIAS

```plaintext
VIAS numVias ;
    [ - viaName
      [ +PATTERNNAME patternName + RECT layerName pt pt]... ; ]...
END VIAS
```

<table>
<thead>
<tr>
<th>DEF</th>
<th>dfII</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIAS numVias</td>
<td>Number of via master cellviews and</td>
</tr>
<tr>
<td></td>
<td>parameterized vias.</td>
</tr>
<tr>
<td>viaName</td>
<td>Cell name of via master cellview.</td>
</tr>
<tr>
<td>PATTERNNAME</td>
<td>Stored as the gec3Param property.</td>
</tr>
<tr>
<td>RECT layerName (pt pt)</td>
<td>Via geometry.</td>
</tr>
</tbody>
</table>

**File – Export – DEF** recreates the VIAS statement based on the via instances placed in the design. If the via instance has a rotation other than 0 degrees, a new via named `<viaName>_<_orientation>` is created with rotated geometries. Here, `<viaName>` is the name of the via master in the database, and `<orientation>` is the orientation of the new via. This methodology is used to support rotation or mirroring of vias. You can add a new via in the technology library, if required.

**Note:** While exporting DEF, only the vias used in the design are written in the DEF file.

### REGIONS Section

The REGIONS section in the DEF file defines the placement regions. These map to placement regions in dfII.

```plaintext
REGIONS numRegions;
    [- regionName pt pt [pt pt]...]
    [+ TYPE { FENCE | GUIDE }]
    [+ PROPERTY {propName propVal}...]...];...]
END REGIONS
```

<table>
<thead>
<tr>
<th>DEF</th>
<th>dfII</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGIONS numRegions</td>
<td>Number of regions in the design.</td>
</tr>
<tr>
<td>regionName</td>
<td>Region name. Stored as the regionName</td>
</tr>
<tr>
<td></td>
<td>property.</td>
</tr>
<tr>
<td>pt pt</td>
<td>Defines region boundaries.</td>
</tr>
<tr>
<td>TYPE { FENCE</td>
<td>GUIDE }</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: While exporting DEF, only the vias used in the design are written in the DEF file.
**Note: DEF version 5.5:** If a region in a design cellview does not have an associated region name, the `regionName` statement is dropped from the `REGIONS` section in the DEF file and a warning message is generated. Whether or not a region has a name, `numRegions` reflects all the regions included in the `REGIONS` section.

For example, there are two regions in a given design cellview but the first region does not have a name. The name of the second region is `region2`. Therefore, the value of `numRegions` in this case is 2. The first region does not have an associated name and is therefore, not printed in the `REGIONS` section. The second region has the name `region2`.

```
REGIONS 2 ;
- region2 ( 3310 660 ) ( 10390 6720 ) ;
END REGIONS
```

**COMPONENTS Section**

The `COMPONENTS` section in the DEF file contains component placement data along with optional netlist data.

```
COMPONENTS numComps ;
[ - compName modelName [netName | *]...]
[ + EEQMASTER macroname ]
[ + GENERATE generatorName [parameters] ]
[ + SOURCE { NETLIST | DIST | USER | TIMING} ]
[ + FOREIGN foreignCellName ( x y ) ori ]...
[ + { FIXED | COVER | PLACED | UNPLACED } ( x y ) ori ]
[ + WEIGHT weight ]
[ + REGION { [ x y ] ( x y ) | regionName} ]
[+ PROPERTY { propName propVal}...]; ]...
END COMPONENTS
```

*File – Import – DEF and File – Export – DEF* do not transfer netlist data to or from the `COMPONENTS` section. Netlist data in the dfII cellview maps to the `NETS` and `SPECIALNETS` sections in DEF. *File – Import – DEF* places each component in the list, with `compName` as the instance name at the given location with the specified orientation and placement status. *File – Export – DEF* generates the cell list based on the location, orientation, and
placement status of the instances in the design. *File – Import – DEF* and *File – Export – DEF* do not support GENERATE.

<table>
<thead>
<tr>
<th>DEF</th>
<th>dfll</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPONENTS</td>
<td>If numComps does not match the number of components in the cell list, <em>File – Import – DEF</em> issues a warning. <em>File – Export – DEF</em> recomputes numComps based on the number of instances in the design.</td>
</tr>
<tr>
<td>compName</td>
<td>Instance name.</td>
</tr>
<tr>
<td>modelName</td>
<td>Cell name of the master cellview for an instance.</td>
</tr>
<tr>
<td>EEQMASTER</td>
<td>Stored as the eeqMaster instance property.</td>
</tr>
<tr>
<td>macroName</td>
<td>Not supported.</td>
</tr>
<tr>
<td>SOURCE</td>
<td>Stored as the source instance property.</td>
</tr>
<tr>
<td>FOREIGN</td>
<td>Stored as the gec3Foreign property of the master cellview.</td>
</tr>
<tr>
<td>foreignCellName</td>
<td>(x y)</td>
</tr>
<tr>
<td>orient</td>
<td>Instance with the firm placement status.</td>
</tr>
<tr>
<td>FIXED</td>
<td>Instance with the locked placement status.</td>
</tr>
<tr>
<td>COVER</td>
<td>Instance with the placed placement status.</td>
</tr>
<tr>
<td>PLACED</td>
<td>Instance with the unplaced placement status.</td>
</tr>
<tr>
<td>UNPLACED</td>
<td>Instance with the firm placement status.</td>
</tr>
<tr>
<td>(x y)</td>
<td>x and y coordinates of the instance origin.</td>
</tr>
<tr>
<td>orient</td>
<td>Orientation of the instance:</td>
</tr>
<tr>
<td></td>
<td>N</td>
</tr>
<tr>
<td></td>
<td>W</td>
</tr>
<tr>
<td></td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>E</td>
</tr>
<tr>
<td></td>
<td>FN</td>
</tr>
<tr>
<td></td>
<td>FW</td>
</tr>
<tr>
<td></td>
<td>FS</td>
</tr>
<tr>
<td></td>
<td>FE</td>
</tr>
<tr>
<td>weight</td>
<td>This statement is ignored.</td>
</tr>
<tr>
<td>REGION</td>
<td>The dfll region with the placement class matching the instance placement class.</td>
</tr>
</tbody>
</table>
Note: DEF version 5.5: If a region in a design cellview does not have an associated region name, the REGION statement is dropped from the COMPONENTS section in the DEF file. For example, in the following COMPONENTS section, the I12 instance is associated with the region region1 and the I13 instance is associated with the region region2.

COMPONENTS 12 ;
   - I12 NAND3_B + PLACED (11159 6359) FN + REGION region1 ;
   - I6 NAND3_B + PLACED (8279 28458) N ;
   - I13 NAND2_C + PLACED (10919 9106) N + REGION region2 ;
END COMPONENTS

If, however, the region to which I13 is associated did not have a name, there would be no REGION statement for the definition of that instance. The COMPONENTS section in that case would be:

COMPONENTS 12 ;
   - I12 NAND3_B + PLACED (11159 6359) FN + REGION region1 ;
   - I6 NAND3_B + PLACED (8279 28458) N ;
   - I13 NAND2_C + PLACED (10919 9106) N ;
END COMPONENTS

PINS Section

The PINS section defines hierarchical I/O pins. A DEF file can have more than one physical pin on a net.

PINS numPins ;
   [ - pinName + NET netName
      [+ SPECIAL]
      [+ DIRECTION {INPUT | OUTPUT | INOUT | FEEDTHRU}]
      [+ USE [SIGNAL | POWER | GROUND | CLOCK | TIEOFF | SCAN | RESET ANALOG]]
      [+ {FIXED | PLACED | COVER} pt orient]
      [+ LAYER layerName pt pt]
      [+ ANTENNAMODEL OXIDE{1,2,3,4}]
      [+ ANTENNAPINPARTIALMETALAREA value [LAYER layerName]]
      [+ ANTENNAPINPARTIALMETALSIDEAREA value [LAYER layerName]]
      [+ ANTENNAPINGATEAREA value [LAYER layerName]]
      [+ ANTENNANAPINDIFFAREA value [LAYER layerName]]
      [+ ANTENNAPINMAXAREAACAR value LAYER layerName]
      [+ ANTENNAPINMAXSIDEAREACAR value LAYER layerName]
      [+ ANTENNAPINPARTIALCUTAREA value [LAYER cutlayerName]]
...
## Design Data Translator's Reference
### Translating LEF and DEF Files

<table>
<thead>
<tr>
<th>DEF</th>
<th>dfII</th>
</tr>
</thead>
<tbody>
<tr>
<td>PINS numPins</td>
<td>Number of hierarchical terminals.</td>
</tr>
<tr>
<td>pinName</td>
<td>Pin name.</td>
</tr>
<tr>
<td>NET netName</td>
<td>Net name, terminal name.</td>
</tr>
<tr>
<td>SPECIAL</td>
<td>Not supported.</td>
</tr>
<tr>
<td>DIRECTION</td>
<td>Terminal direction:</td>
</tr>
<tr>
<td>INPUT</td>
<td>Input</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>Output</td>
</tr>
<tr>
<td>INOUT</td>
<td>InputOutput</td>
</tr>
<tr>
<td>FEEDTHRU</td>
<td>Jumper</td>
</tr>
<tr>
<td>USE</td>
<td>Signal type of net signal:</td>
</tr>
<tr>
<td>SIGNAL</td>
<td>signal</td>
</tr>
<tr>
<td>POWER</td>
<td>supply</td>
</tr>
<tr>
<td>GROUND</td>
<td>ground</td>
</tr>
<tr>
<td>CLOCK</td>
<td>clock</td>
</tr>
<tr>
<td>USE</td>
<td>use property of net:</td>
</tr>
<tr>
<td>TIEOFF</td>
<td>“tieoff”</td>
</tr>
<tr>
<td>ANALOG</td>
<td>“analog”</td>
</tr>
<tr>
<td>SCAN</td>
<td>“scan”</td>
</tr>
<tr>
<td>RESET</td>
<td>“reset”</td>
</tr>
<tr>
<td>FIXED</td>
<td>Terminal pin with fixed status.</td>
</tr>
<tr>
<td>PLACED</td>
<td>Terminal pin with placed status.</td>
</tr>
<tr>
<td>COVER</td>
<td>Terminal pin with cover status.</td>
</tr>
<tr>
<td>pt</td>
<td>x and y coordinates of terminal pin origin.</td>
</tr>
<tr>
<td>FIXED</td>
<td>Terminal pin with firm placement status.</td>
</tr>
<tr>
<td>PLACED</td>
<td>Terminal pin with placed placement status.</td>
</tr>
<tr>
<td>COVER</td>
<td>Terminal pin with locked placement status.</td>
</tr>
</tbody>
</table>
## DEF

<table>
<thead>
<tr>
<th><strong>DEF</strong></th>
<th><strong>dfll</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>orient</strong></td>
<td>Orientation of terminal pin:</td>
</tr>
<tr>
<td>N</td>
<td>r0</td>
</tr>
<tr>
<td>W</td>
<td>r90</td>
</tr>
<tr>
<td>S</td>
<td>r180</td>
</tr>
<tr>
<td>E</td>
<td>r270</td>
</tr>
<tr>
<td>FN</td>
<td>my</td>
</tr>
<tr>
<td>FW</td>
<td>myr90</td>
</tr>
<tr>
<td>FS</td>
<td>mx</td>
</tr>
<tr>
<td>FE</td>
<td>mxr270</td>
</tr>
<tr>
<td><strong>LAYER layerName pt pt</strong></td>
<td>Pin is created on the layer <em>layerName</em> and geometry <em>pt pt</em>.</td>
</tr>
<tr>
<td><strong>ANTENNAMODEL OXIDE</strong></td>
<td>Mapped as a hierarchical property. Supported values are 1 and 2.</td>
</tr>
<tr>
<td><strong>ANTENNAPINPARTIALMETALAREA</strong> value [LAYER layerName]**</td>
<td>Mapped as a hierarchical property. The parent property name is antennaPinPartialMetalArea. The hierarchical property name is layerName or DEFAULT (if layerName is not specified), the hierarchical property type is integer, and the hierarchical property value is value/DEFConvertFactor^2.</td>
</tr>
<tr>
<td><strong>ANTENNAPINPARTIALMETALSIDE AREA</strong> value [LAYER layerName]**</td>
<td>Mapped as a hierarchical property. The parent property name is antennaPinPartialMetalSideArea. The hierarchical property name is layerName or DEFAULT (if layerName is not specified), the hierarchical property type is integer, and the hierarchical property value is value/DEFConvertFactor^2.</td>
</tr>
<tr>
<td><strong>ANTENNAPINGATEAREA</strong> value [LAYER layerName]**</td>
<td>Mapped as a child property of ANTENNAMODEL OXIDE (1 and 2) with property name as antennaPinGateArea. The hierarchical property of antennaPinGateArea is layerName or DEFAULT (if layerName is not specified), the hierarchical property type is integer, and the hierarchical property value is value/DEFConvertFactor^2.</td>
</tr>
</tbody>
</table>
### Design Data Translator's Reference
#### Translating LEF and DEF Files

<table>
<thead>
<tr>
<th>DEF</th>
<th>dfII</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANTENNAPINDIFFAREA value [LAYER layerName]</td>
<td>Mapped as a hierarchical property. The parent property name is antennaPinDiffArea. The hierarchical property name is layerName or DEFAULT (if layerName is not specified), the hierarchical property type is integer, and the property value is value/DEFConvertFactor^2.</td>
</tr>
<tr>
<td>ANTENNAPINMAXAREACAR value LAYER layerName</td>
<td>Mapped as a child property of on a pinANTENNAOxide (1 and 2) with property name as antennaPinMaxAreaCar. The hierarchical property name of antennaPinMaxAreaCar is layerName, the property type is integer, and the property value is value/DEFConvertFactor^2.</td>
</tr>
<tr>
<td>ANTENNAPINMAXSIDEAREACAR value LAYER layerName</td>
<td>Mapped as a child property of ANTENNAOxide (1 and 2) with property name as antennaPinMaxSideAreaCar. The hierarchical property of antennaPinMaxSideAreaCar is layerName, the property type is integer, and the property value is value/DEFConvertFactor^2.</td>
</tr>
<tr>
<td>ANTENNAPINPARTIALCUTAREA value [LAYER cutlayerName]</td>
<td>Mapped as a hierarchical property. The parent property name is antennaPinPartialCutArea. The hierarchical property name is cutlayerName or DEFAULT (if cutlayerName is not specified), the hierarchical property type is integer, and the property value is value/DEFConvertFactor^2.</td>
</tr>
</tbody>
</table>
All the antenna properties are stored as child properties of ANTENNAMODEL OXIDE. All layers on these antenna properties are also stored as child properties of the ANTENNAMODEL OXIDE property.

If there is no antenna model specified, OXIDE1 is considered as the default value. All the antenna properties for OXIDE3 and OXIDE4 will be ignored and following warning message will be displayed:

*WARNING* All ANTENNA properties for ANTENNAMODEL %s will be ignored.

DEF Out will print the ANTENNAMODEL OXIDE property only if the version is 5.5.

Antenna rules for a pin are mapped as hierarchical properties on the terminal pin.

In DEF 5.4, ANTENNA*AREA units are integers. These units are measured as distance in database units squared.

Example

```
UNIT DISTANCE MICRON 1000;
PIN
.........
+ ANTENNAPINPARTIALMETALAREA 2000000;
....
```

In the above statement, the value of one DB unit is 1000. Therefore, the area value unit is 1000000 (1000^2). The value for ANTENNAPINPARTIALMETALAREA is 2um^2 (2000000/1000000).

In all the antenna rules, due to data overflow, you can specify value up to 2,147,483,647 database units. In microns squared, this is 2,147,483,647/DEFConvertFactor^2.

If the value of USE is SIGNAL, it will not be written to the output file. By default, the value of USE is SIGNAL.

DEF file can have more than one physical PIN on a net.
In the design of place and route blocks, you might want to add extra physical connection points to existing signal ports (usually to enable the signal to be accessed from two sides of the block). One pin has the same name as the net to which it is connected. This pin is related to the pins specified in GCF for boundary conditions. Any other pins added to the net must use the naming conventions described below:

- For extra non-bus bit pin names, use the following syntax:
  \[\text{pinname}\.extraN\]
  where \(N\) is a positive integer that is incremented as the physical pins are added.

**Example**

\[
\begin{align*}
\text{PINS n ;} \\
&- \text{a + NET a .... ;} \\
&- \text{a.extral + NET a ... ;}
\end{align*}
\]

- For extra bus bit pin names, use the following syntax:
  \[\text{basename}\.extraN[index]\]

where,

- \textit{basename}\hspace{2cm} Simple part of the bus bit pin or net name
- \(N\) \hspace{2cm} A positive integer, incremented as the physical pins are added.
- \([index]\) \hspace{2cm} Identifies a specific bit of the bus

**Example**

\[
\begin{align*}
\text{PINS n ;} \\
&- \text{a[0] + net a[0] ... ;} \\
&- \text{a.extral[0] + net a[0] ... ;}
\end{align*}
\]

**Note:** The square brackets are \texttt{BUSBITCHARS} values as defined in the DEF header.

**IOTimings Section**

The \texttt{IOTIMINGS} section of the DEF file contains I/O constraints used to set boundary conditions for a block or chip in delay calculation.
This section was supported only up to DEF version 5.3.

IOTIMINGS numIOTimings;
    [ - (comp pin) | (PIN name)]
    [ + { RISE | FALL } VARIABLE min max ]...
    [ + { RISE | FALL } SLEWRATE min max ]...
    [ + CAPACITANCE capacitance ]
    [ + DRIVECELL macroname [[FROMPIN pinname] TOPIN pinname]
        [PARALLEL ndr...]]
    ; ]...
ENDIOTIMINGS

If DRIVECELL is present in the DEF file, File – Import – DEF writes RISE SLEWRATE and FALL SLEWRATE as GCF data. If DRIVECELL is not present, File – Import – DEF writes RISE SLEWRATE and FALL SLEWRATE as ISPF data.

<table>
<thead>
<tr>
<th>DEF</th>
<th>dfII</th>
</tr>
</thead>
<tbody>
<tr>
<td>RISE SLEWRATE</td>
<td>(ISPF)</td>
</tr>
<tr>
<td>min</td>
<td>adppaRiseSlewRateMin property of I/O terminal</td>
</tr>
<tr>
<td>max</td>
<td>adppaRiseSlewRateMax property of I/O terminal</td>
</tr>
<tr>
<td></td>
<td>(GCF)</td>
</tr>
<tr>
<td>min</td>
<td>adpDCRiseSlewRateMin property of I/O terminal</td>
</tr>
<tr>
<td>max</td>
<td>adpDCRiseSlewRateMax property of I/O Terminal</td>
</tr>
<tr>
<td>FALL SLEWRATE</td>
<td>(ISPF)</td>
</tr>
<tr>
<td>min</td>
<td>adppaFallSlewRateMin property of I/O terminal</td>
</tr>
<tr>
<td>max</td>
<td>adppaFallSlewRateMax property of I/O terminal</td>
</tr>
<tr>
<td></td>
<td>(GCF)</td>
</tr>
<tr>
<td>min</td>
<td>adpDCFallSlewRateMin property of I/O terminal</td>
</tr>
<tr>
<td>max</td>
<td>adpDCFallSlewRateMax property of I/O terminal</td>
</tr>
<tr>
<td>CAPACITANCE</td>
<td>adppaCap property of I/O terminal</td>
</tr>
<tr>
<td>capacitance</td>
<td></td>
</tr>
<tr>
<td>RISE VARIABLE</td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>adppaRiseVariableMin property of I/O terminal</td>
</tr>
<tr>
<td>max</td>
<td>adppaRiseVariableMax property of I/O terminal</td>
</tr>
<tr>
<td>FALL VARIABLE</td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>adppaFallVariableMin property of I/O terminal</td>
</tr>
<tr>
<td>max</td>
<td>adppaFallVariableMax property of I/O terminal</td>
</tr>
</tbody>
</table>
**Caution**

*DEF inputs the IOTIMING properties of PINS on the I/O terminals in dfII from the input DEF file. DEF Out writes these properties for all the I/O pins in dfII by reading them from their respective terminals as per their connectivity in the dfII database.*


**NETS Section**

The NETS section in the DEF file contains the netlist data along with optional routing information for nets. Pins specified in the NETS section are regular pins. Nets specified in the NETS section are routed by the global and final routers. DEF uses the SPECIALNETS section to describe the nets connected to special pins, which are routed by a special router.
If the NETS section specifies a non-default rule for a net, the routers derive the wiring width and vias for the net from the non-default rule. If a net has no non-default rule, the routers use default-width wiring and default vias.

```
NETS numNets;
    [- ( netName [{ (compName | PIN) pinName[+ SYNTHESIZED]}]... |
        MUSTJOIN (compName pinName) )
    [+ SHIELDNET shieldNetName]
    [+ VPIN vpiname [LAYER layerName pt pt]
        [ { PLACED | FIXED | COVER } pt orient ] ]...
    [+ SUBNET subnetName
        [ (compName pinName) | (VPIN vpiname) ]...
        [NONDEFAULTRULE rulename]
        [regular wiring]]
    [+ XTALK num]
    [NONDEFAULTRULE ruleName]
    [+ regularWiring ]...
    [+ SOURCE { NETLIST | DIST | USER | TEST | TIMING } ]
    [+ FIXEDBUMP]
    [+ FREQUENCY]
    [+ ORIGINAL netName ]
    [+ USE
        { SIGNAL | POWER | GROUND | CLOCK | SCAN | RESET
            | TIEOFF | ANALOG } ]
    [+ PATTERN
        { STEINER | BALANCED | WIREDLOGIC } ]
    [+ ESTCAP wireCapacitance ]
    [+ WEIGHT weight ]
    [+ PROPERTY { propName propVal}...]]...; ]...
END NETS
```

```
regularWiring=(ROUTED | FIXED | COVER | NOSHIELD)
layerName [TAPER]
(x y) [(x*) | (*y) | viaName]
[NEW layerName
(x y) [(x*) | (*y) | viaName]]
```

*File – Import – DEF* and *File – Export – DEF* transfer netlist connectivity, and full or partial wiring for nets.

<table>
<thead>
<tr>
<th>DEF</th>
<th>dfll</th>
</tr>
</thead>
</table>

```
netName Name of the net.
```
### DEF

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compName pinName</td>
<td>Instance terminals attached to the net.</td>
</tr>
<tr>
<td>PIN pinName</td>
<td>Terminal attached to net. To get an output of all the terminals on the net, set the value of the SKILL variable OUTPUT_DEF_IOPINSINNETSSECTION to “TRUE”. By default, the value of OUTPUT_DEF_IOPINSINNETSSECTION is “FALSE”.</td>
</tr>
<tr>
<td>SYNTHESIZED</td>
<td>gec3Synthesized property of the instance terminal.</td>
</tr>
<tr>
<td>SHIELDNET shieldNetName</td>
<td>A special net that shields the current regular net netname. Maps as a value of the property SHIELDNET on this net. The value of SHIELDNET is a list containing all the special nets specified in the SHIELDNET statement for a regular net.</td>
</tr>
<tr>
<td>MUSTJOIN(compName pinName)</td>
<td>Routed mustjoin connection.</td>
</tr>
<tr>
<td>XTALK</td>
<td>Stored as the XTALK property on the net.</td>
</tr>
<tr>
<td>VPIN vpinName</td>
<td>Pin on netName. Physical information and status of the pin are mapped as for I/O pins.</td>
</tr>
<tr>
<td>SUBNET subnetName</td>
<td>Subnet of the net netname.</td>
</tr>
<tr>
<td>VPIN vpinName</td>
<td>Property of subnetName.</td>
</tr>
<tr>
<td>NONDEFAULTRULE gec3NondefaultRuleName</td>
<td>gec3NondefaultRuleName property of the net (name of nondefault rule for regular wiring for net).</td>
</tr>
<tr>
<td>SOURCE</td>
<td>Stored as the source property of the net.</td>
</tr>
<tr>
<td>FIXEDBUMP</td>
<td>Stored as the boolean property fixedBump on the net.</td>
</tr>
<tr>
<td>FREQUENCY</td>
<td>Stored as the float property frequency on the net.</td>
</tr>
<tr>
<td>ORIGINAL netName</td>
<td>Stored as the original property of the net.</td>
</tr>
<tr>
<td>DEF</td>
<td>dfII</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>USE</td>
<td>Signal type of net signal:</td>
</tr>
<tr>
<td></td>
<td>- SIGNAL: signal</td>
</tr>
<tr>
<td></td>
<td>- POWER: supply</td>
</tr>
<tr>
<td></td>
<td>- GROUND: ground</td>
</tr>
<tr>
<td></td>
<td>- CLOCK: clock</td>
</tr>
<tr>
<td></td>
<td>USE property of net:</td>
</tr>
<tr>
<td></td>
<td>- TIEOFF: “tieoff”</td>
</tr>
<tr>
<td></td>
<td>- ANALOG: “analog”</td>
</tr>
<tr>
<td></td>
<td>- SCAN: “scan”</td>
</tr>
<tr>
<td></td>
<td>- RESET: “reset”</td>
</tr>
<tr>
<td>PATTERN</td>
<td>Stored as the pattern property of the net.</td>
</tr>
<tr>
<td>ESTCAP</td>
<td>estCap property of the net.</td>
</tr>
<tr>
<td>WEIGHT weight</td>
<td>prEditNetPriority property of net. Net weights in dfII can have values from 0 to 127. File – Export – DEF adjusts these values into a 1 to 128 range.</td>
</tr>
<tr>
<td>PROPERTY propName propVal</td>
<td>Stored as the hierarchical property on the net. The parent property name is DEFREGNETHIERPROP and the hierarchical property name is propName.</td>
</tr>
<tr>
<td>regularWiring</td>
<td>Provides the geometry of the net.</td>
</tr>
<tr>
<td>ROUTED</td>
<td>Specifies that the wiring can be moved by the automatic layout tools. The defstatus property is not set.</td>
</tr>
<tr>
<td>FIXED</td>
<td>Mapped as the value of the property defstatus on the corresponding path on the net. The value of defstatus is firm.</td>
</tr>
<tr>
<td>COVER</td>
<td>Mapped as the value of the property defstatus on the corresponding path for the net. The value of defstatus is locked.</td>
</tr>
<tr>
<td>NOSHIELD</td>
<td>Mapped as boolean property defshieldStatus on the corresponding path for the net. The value of defshieldStatus property is true.</td>
</tr>
</tbody>
</table>
Regular Wiring

The regular wiring for a net is the physical routing that routes to regular pins. Pins specified in the NETS section are regular pins. Wiring generated by the global, quick, and final routers is always regular. Regular wiring has the following DEF format:

\[
\{ \text{ROUTED} \mid \text{FIXED} \mid \text{COVER} \} \text{layerName} [\text{TAPER}]
\]
\[
( x \ y ) [ ( x \ast ) \mid ( \ast \ y ) \mid \text{viaName} ]...
\]

[ NEW \text{layerName}
\]
\[
( x \ y ) [ ( x \ast ) \mid ( \ast \ y ) \mid \text{viaName} ]...
\]

File – Import – DEF maps the wiring for a net to geometric objects attached to the net as follows:

- Vias map to via instances.
- Wires map to path geometry objects with pathStyle set to extendExtend.

Note: File – Import – DEF associates instance terminals with nets, deriving the pin names from master cell abstract cellviews. For File – Import – DEF to succeed, port names in the DEF file must match the pin names in the master cellviews in the cell library.

File – Import – DEF also sets the net property sroute. If File – Import – DEF finds all the wiring for a routed or partially routed net in the NETS section of the DEF file, the command creates the net with the sroute property set to none. If File – Import – DEF finds routing for a net in both the NETS and SPECIALNETS sections of the DEF file, the command creates the net with the sroute property set to some.

The dfII tools use the sroute property, along with wire width, to determine whether File – Export – DEF describes a wire segment or via in the NETS or the SPECIALNETS section of the DEF file.

If the sroute property of a net is all, File – Export – DEF describes the net and its routing in the SPECIALNETS section. If the value of the sroute property for a net is some or none, File – Export – DEF describes wires with the default routing width as regular default routing in the NETS section. Instances of default vias also appear as regular default routing. Vias named in the nondefault rule for the net and wires with a width equal to the nondefault rule width appear in the NETS section as nondefault regular routing.

If the sroute property is set to some, File – Export – DEF describes wires with widths other than the default or specified nondefault widths in the SPECIALNETS section as special net routing. Nondefault vias not named in the nondefault rule for the net also appear in the SPECIALNETS section. If the sroute property is none, File – Export – DEF treats all wires not at the default width as nondefault regular wiring.

File-Export – DEF writes vias with a new name if the instance is a rotated or mirrored version of the library via. The generated name is <viaName>_<orientation>, where
<viaName> is the name of the via master in the database, and <orientation> is the orientation of via instance in the design cellview.

<table>
<thead>
<tr>
<th>DEF</th>
<th>dfll</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROUTED</td>
<td>Wire with routed status.</td>
</tr>
<tr>
<td>FIXED</td>
<td>Wire with firm status.</td>
</tr>
<tr>
<td>COVER</td>
<td>Wire with locked status.</td>
</tr>
</tbody>
</table>

layerName

Endpoints coordinates of wire segments. Wires have the nondefault width specified for the net, if any. Otherwise, the wires have the default width specified for the layer.

TAPER coordinates

Wire segments with default width for layer.

viaName

Via instance placed at last coordinates. File – Export – DEF treats vias listed in the nondefault rule for the net as nondefault routing.

NEW coordinates

Discontinuous wire segment. If a nondefault rule is specified for the net, the wire has nondefault width.

Note: During DEF Out, special vias are no longer dropped from the regularWiring section. Whenever special vias are encountered in a regular net, following warning message is displayed, but special via is retained in the NETS section.

Regular Net <name> contains special vias

SPECIALNETS Section

Like the NETS section, the SPECIALNETS section in the DEF file contains netlist data plus optional routing information for the nets. Pins specified in the SPECIALNETS section are special pins. The special router routes to these pins using wide wiring and special or default vias. DEF 5.3 onwards, shielded nets are supported in the specialWiring statement. The
specialWiring statement describes the geometry of a special net. For more information on the specialWiring syntax, see the description of Regular Wiring in the NETS section.

SPECIALNETS numNets ;

    [- netName [ (compRegExpr pinName) ]... Not supported in DEF 5.5
        [ + WIDTH layerName width ]... Supported in DEF 5.2
        [ + VOLTAGE volts ]
        [ + SPACING layerName spacing ]...
        [ + specialWiring ]...
        [ + SOURCE { NETLIST | DIST | USER | TIMING } ]
        [ + FIXEDBUMP]
        [ + ORIGINAL netName ]
        [ + USE
            (SIGNAL | POWER | GROUND | CLOCK | SCAN | RESET
             | TIEOFF | ANALOG) ]
        [ + PATTERN
            { STEINER | BALANCED | WIREDLOGIC | TRUNK } ]
        [ + ESTCAP wireCapacitance ]
        [ + WEIGHT weight ]
        [+ PROPERTY {propName propVal}...]; ]...

END SPECIALNETS

specialWiring =

    {ROUTED | FIXED | COVER | SHIELD shieldedNetName}

    layerName width

    [+ SHAPE {RING | STRIPE | FOLLOWPIN | IO_WIRE | CORE_WIRE | PADRING | BLOCKRING
               BLOCKWIRE | FILLWIRE | BLOCKAGEWIRE | DRCFILL}]

    (x y) [(x*) | (*y) | viaName [DO num X by Y STEP stepX stepY]]...

    [NEW layerName width

    [+ SHAPE {RING | STRIPE | FOLLOWPIN | IO_WIRE | CORE_WIRE |
               BLOCKWIRE | FILLWIRE | BLOCKAGEWIRE | DRCFILL}]

    (x y) [(x*) | (*y) | viaName [DO num X by Y STEP stepX stepY]]...

---

**DEF**

<table>
<thead>
<tr>
<th>numNets</th>
<th>Number of nets in the design. File – Import – DEF checks for valid numNets.</th>
</tr>
</thead>
<tbody>
<tr>
<td>netName</td>
<td>Name of the net.</td>
</tr>
<tr>
<td>compRegExpr</td>
<td>Instance terminals attached to the net.</td>
</tr>
<tr>
<td>pinName</td>
<td></td>
</tr>
<tr>
<td>WIDTH layerName width</td>
<td>Width of wires on layer for the net.</td>
</tr>
<tr>
<td>VOLTAGExx</td>
<td>Not supported.</td>
</tr>
<tr>
<td>DEF</td>
<td>dfll</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>SPACING</td>
<td>specialSpacing property of the net.</td>
</tr>
<tr>
<td>specialWiring</td>
<td></td>
</tr>
<tr>
<td>ROUTED</td>
<td>Specifies that the wiring can be moved by the automatic layout tools. The defstatus property is not set.</td>
</tr>
<tr>
<td>FIXED</td>
<td>Mapped as the value of the property defstatus on the corresponding path on the net. The value of defstatus is firm.</td>
</tr>
<tr>
<td>COVER</td>
<td>Mapped as the value of the property defstatus on the corresponding path on the net. The value of defstatus is locked.</td>
</tr>
<tr>
<td>SHIELD</td>
<td>Mapped as the property shield on the corresponding path for the net.</td>
</tr>
<tr>
<td>layerName</td>
<td>Mapped as the property shapeProp.</td>
</tr>
<tr>
<td>SHAPE</td>
<td></td>
</tr>
<tr>
<td>RING</td>
<td>“RING”</td>
</tr>
<tr>
<td>STRIPE</td>
<td>“STRIPE”</td>
</tr>
<tr>
<td>FOLLOWPIN</td>
<td>“FOLLOWPIN”</td>
</tr>
<tr>
<td>IOWIRE</td>
<td>“IOWIRE”</td>
</tr>
<tr>
<td>COREWIRE</td>
<td>“COREWIRE”</td>
</tr>
<tr>
<td>PADRING</td>
<td>“PADRING”</td>
</tr>
<tr>
<td>BLOCKRING</td>
<td>“BLOCKRING”</td>
</tr>
<tr>
<td>BLOCKWIRE</td>
<td>“BLOCKWIRE”</td>
</tr>
<tr>
<td>FILLWIRE</td>
<td>“FILLWIRE”</td>
</tr>
<tr>
<td>BLOCKAGEWIRE</td>
<td>“BLOCKAGEWIRE”</td>
</tr>
<tr>
<td>DRCFILL</td>
<td>“DRCFILL”</td>
</tr>
<tr>
<td>x*</td>
<td>Indicates that the y coordinate last specified in the wiring specification is used.</td>
</tr>
<tr>
<td>y*</td>
<td>Indicates that the x coordinate last specified is used.</td>
</tr>
<tr>
<td>NEW</td>
<td>Specifies that there is no wire segment between the coordinate specified last and the next coordinate.</td>
</tr>
<tr>
<td>viaName</td>
<td>Specifies that a via is placed at the last specified coordinate.</td>
</tr>
<tr>
<td>DO num X by Y STEP</td>
<td>Via array will be converted to individual vias and written as single vias during DEF Out.</td>
</tr>
<tr>
<td>stepX stepY</td>
<td></td>
</tr>
</tbody>
</table>
**File – Import – DEF** maps special routing data the same way as regular wiring, except wire widths are explicit and vias can be nondefault. In addition, **File – Export – DEF** generates the spacing statements from a list of `layerName-spacing` pairs that make up the `specialSpacing` property of the net instance. The shape of a wire is stored as a string property.

**File – Import – DEF** also sets the net property `sroute`. If **File – Import – DEF** finds all the wiring for a routed or partially routed net in the `SPECIALNETS` section of the DEF file, the command creates a net with the `sroute` property set to `all`. If **File – Import – DEF** finds routing for a net in both the `NETS` and `SPECIALNETS` sections of the DEF file, the command creates a net with the `sroute` property set to `some`.

The dfII tools use the net property `sroute`, along with wire width, to determine whether **File – Export – DEF** describes a wire segment or via in the `NETS` or the `SPECIALNETS` sections of the DEF file.

<table>
<thead>
<tr>
<th>DEF</th>
<th>dfII</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOURCE</td>
<td>Stored as the <code>source</code> property of net.</td>
</tr>
<tr>
<td>FIXEDBUMP</td>
<td>Stored as the boolean property <code>fixedBump</code> on the net.</td>
</tr>
<tr>
<td>ORIGINAL <code>netName</code></td>
<td>Stored as the <code>originalName</code> property of net.</td>
</tr>
<tr>
<td>USE <code>netName</code></td>
<td>Signal type of net signal:</td>
</tr>
<tr>
<td>SIGNAL</td>
<td><code>signal</code></td>
</tr>
<tr>
<td>POWER</td>
<td><code>supply</code></td>
</tr>
<tr>
<td>GROUND</td>
<td><code>ground</code></td>
</tr>
<tr>
<td>CLOCK</td>
<td><code>clock</code></td>
</tr>
<tr>
<td>USE <code>use</code></td>
<td><code>use</code> property of net:</td>
</tr>
<tr>
<td>TIEOFF</td>
<td>&quot;tieoff&quot;</td>
</tr>
<tr>
<td>ANALOG</td>
<td>&quot;analog&quot;</td>
</tr>
<tr>
<td>SCAN</td>
<td>&quot;scan&quot;</td>
</tr>
<tr>
<td>RESET</td>
<td>&quot;reset&quot;</td>
</tr>
<tr>
<td>PATTERN</td>
<td>Stored as the <code>pattern</code> property of net.</td>
</tr>
<tr>
<td>ESTCAP</td>
<td><code>maxCap</code> property of the net.</td>
</tr>
<tr>
<td>WEIGHT <code>weight</code></td>
<td><code>prEditNetPriority</code> property of the net. Net weights in dfII can have values from 0 to 127. <strong>File – Export – DEF</strong> adjusts these values into a 1 to 128 range.</td>
</tr>
<tr>
<td>PROPERTY <code>propName</code></td>
<td>List of properties set on a net object in the database. Here, <code>propName</code> is the name of the property, and <code>propVal</code> is the value of the property.</td>
</tr>
</tbody>
</table>

---

*Source:* Design Data Translator's Reference | Translating LEF and DEF Files | July 2007 409 Product Version 5.1.41
If the `sroute` property of a net is *all*, *File – Export – DEF* describes the net and its routing in the `SPECIALNETS` section. If the value of the `sroute` property for a net is *some* or *none*, *File – Export – DEF* describes wires with the default routing width as regular default routing in the `NETS` section. Instances of default vias also appear as regular default routing. Vias named in the nondefault rule for nets and wires with a width equal to the nondefault rule width appear in the `NETS` section as nondefault regular routing.

If the `sroute` property is set to *some*, *File – Export – DEF* describes wires with widths other than the default or specified nondefault widths in the `SPECIALNETS` section as special net routing. Nondefault vias not named in the nondefault rule for the net also appear in the `SPECIALNETS` section. If the `sroute` property is *none*, *File – Export – DEF* treats all wires not at the default width as nondefault regular wiring.

**Note:** The `dfII PR Flatten` and `Generate Physical Hierarchy` commands do not mark nets as special when you use them to create an `autoLayout` cellview. To mark a net as special, use the `Floorplan – Special Route – Set Nets Special` command before you use the *File – Export – DEF* command.

### Support for Arrayed Vias

Up to DEF 5.4, separate instances were created for all the vias in the designs while importing and exporting DEF files. DEF 5.5 now provides the feature of creating mosaics in a design while importing DEF files and arrayed vias while exporting DEF files. The arrayed vias are created as a single object in a design by using mosaics in the database.

A mosaic is a database object that represents an array of instances of one or more masters. There are two kinds of mosaics: simple and complex. A simple mosaic is a special kind of regular array that has only one master, and every cell in the array has the same orientation. Any mosaic other than the simple mosaic is called a complex mosaic. For more information on mosaics, see *Integrator’s Toolkit: Database Reference Manual*.

The type of mosaic instances created when you import and export DEF files using DEF 5.5 are simple mosaics. Following cases provide information about how the arrayed vias get translated while using different versions of DEF.

**Case I**

While importing design files using DEF 5.4, the arrayed vias get converted to separate via instances. If you export files using the same version, the separated via instances remain unchanged.
Case II

While importing design files using DEF 5.4, the arrayed vias change to separate via instances. If you export the files using DEF version 5.5, the separated via instances remain unchanged.

Case III

While importing DEF files using DEF 5.5, the arrayed vias get converted to mosaic instances. If you export DEF files using DEF 5.4, the mosaic instances change to separate via instances.

Case IV

While importing DEF files using DEF 5.5, the arrayed vias get converted to mosaic instances. If you export DEF files using the same version, the mosaic instances remain unchanged.

Note: When the environment variable NOT_USE_RULEVIA is set, DEF In will no longer use the ruleVia to create pcell instances corresponding to the vias encountered in the NETS and SPECIALNETS section. This applies to the design-specific vias (vias defined in DEF but not present in the library).

During DEF In, a search is done for the existence of the vias in technology library and the reference libraries. If not found, it will create them in the form of cellviews in a new library AUTO_DEF_VIAS_LIB. So, these new cellviews will be used while creating via instances during processing of NETS and SPECIALNETS. They will no longer be pcell instances and should also provide massive performance improvement, and reduction in the memory usage.

DEF Out has also been enhanced to recognize the vias defined in AUTO_DEF_VIAS_LIB. So, if an instance of a via master defined in AUTO_DEF_VIAS_LIB is found in the current cellview, then the master via is included in the VIAS section of the DEF file and the wire which contains the instance of the via appears in the NETS or SPECIALNETS section.

Support for Rectangles and Polygons

With the version setting as 5.6, DEF supports the reading and writing of polygons and rectangles in the release IC5.1.41USR3 for the specialWiring statement in the DEF file. If the DEF version is not set to 5.6, the DEF reader and writer will ignore the shapes.

With the DEF version setting as 5.6, the DEF reader processes polygons and rectangles and stores them in the database. Similarly, the DEF writer with the DEF version set to 5.6 writes polygons and rectangles when a DEF version 5.6 output is requested.
This support is available only for the specialWiring statement in DEF version 5.6:

```plaintext
[ + POLYGON layerName pt pt ...
   + RECT layerName pt pt
   (+ ROUTED | + FIXED | + COVER | + SHIELD shieldedNetName)
   layerName routeWidth
   [+ SHAPE {RING | PADRING | BLOCKRING | STRIPE | FOLLOWPIN
            | IOWIRE | COREWIRE | BLOCKWIRE | BLOCKAGEWIRE
            | FILLWIRE | DRCFILL}]
   [+ STYLE styleNum]
   routingPoints
   [NEW layerName routeWidth
    [ + SHAPE {RING | PADRING | BLOCKRING | STRIPE | FOLLOWPIN
              | IOWIRE | COREWIRE | BLOCKWIRE | BLOCKAGEWIRE
              | FILLWIRE | DRCFILL}]
    [+ STYLE styleNum]
    routingPoints
   ]]
][...
][...
```

⚠️**Important**

This is the only support extended by DEF 5.6 in the release IC5.1.41USR3. When the version setting for DEF is 5.6, DEF In and DEF Out generate appropriate warning messages regarding the limitations of the support provided by CDB DEF 5.6 translators.

**GROUPS Section**

Each soft group in DEF directs the placer to cluster components in a group. You use MAXHALFPERIMETER, MAXX, and MAXY to set upper bounds on the size of the area in which the placer can put the members of the group. The optional REGION statement specifies a rectangular region on the design for the group members.

```plaintext
GROUPS numGroups ;
   [ - groupName compNameRegExpr...
      [ + SOFT
         [MAXHALFPERIMETER value]
         [MAXX value][MAXY value] ]
      [ + REGION pt pt | regionName ]
      [+ PROPERTY {propName propVal}...]; ]...
END GROUPS
```

Not supported in DEF 5.5
Soft groups in the DEF file map to dfII groups that have the softGroup property. Members of these groups are cell instances.

<table>
<thead>
<tr>
<th>DEF</th>
<th>dfII</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupName</td>
<td>Name of the soft group.</td>
</tr>
<tr>
<td>compNameRegExp</td>
<td>Member instances of the soft group.</td>
</tr>
<tr>
<td>MAXHALFPERIMETER</td>
<td>softHalfPerimeter property of the group.</td>
</tr>
<tr>
<td>MAXX</td>
<td>softMaxX property of the group.</td>
</tr>
<tr>
<td>MAXY</td>
<td>softMaxY property of the group.</td>
</tr>
<tr>
<td>REGION</td>
<td>Placement region with the same placement class as the group.</td>
</tr>
<tr>
<td>PROPERTY propName propVal</td>
<td>List of properties set on a net object in the database. Here, propName is the name of the property, and propVal is the value of the property.</td>
</tr>
</tbody>
</table>

**CONSTRAINTS Section**

The CONSTRAINTS section was supported only up to DEF version 5.3.

DEF constraints are the upper bounds of specified nets for the following characteristics:

- Delay constraints are the upper limits on the propagation delays (fanout) along the net for rising and falling signals.
- Wired-logic distance constraints are the upper limits on the orthogonal distances between pairs of output pins on a net.

Silicon Ensemble attempts to satisfy these constraints by placing components appropriately using wire-length estimators to calculate delay and distance values.

```
CONSTRAINTS numConstraints ;
   { - { operand
          [ + RISEMAX time ]
          [ + FALLMAX time ]
          [ + RISEMIN time ]
          [ + FALLMIN time ] ;
          | WIREDLOGIC netName MAXDIST distance } ; }...
END CONSTRAINTS
```
Design Data Translator's Reference
Translating LEF and DEF Files

operand =
{ NET netName
 | PATH {comp fromPin | PIN fromPinName} {comp toPin | PIN toPinName}
 | SUM { operand, ... } }


<table>
<thead>
<tr>
<th>DEF</th>
<th>dfll</th>
</tr>
</thead>
<tbody>
<tr>
<td>numConstraints</td>
<td>Number of constraints in design. File – Import – DEF checks for valid numConstraints</td>
</tr>
<tr>
<td>NET netName</td>
<td>Constraint on the net.</td>
</tr>
<tr>
<td>PATH comp fromPin</td>
<td>Constraint on the path. Specifies an instance and instance pin pair as one end of the path on which you put the constraint.</td>
</tr>
<tr>
<td>comp toPin</td>
<td></td>
</tr>
<tr>
<td>PIN fromPinName</td>
<td>An IO pin as one end of the path on which you put the constraint.</td>
</tr>
<tr>
<td>PIN toPinName</td>
<td></td>
</tr>
<tr>
<td>SUM</td>
<td>SUM for PATH is supported.</td>
</tr>
<tr>
<td>RISEMAX</td>
<td>Maximum rise constraint for net or path.</td>
</tr>
<tr>
<td>FALLMAX</td>
<td>Maximum fall constraint for net or path.</td>
</tr>
<tr>
<td>RISEMIN</td>
<td>Minimum rise constraint for net or path.</td>
</tr>
<tr>
<td>FALLMIN</td>
<td>Minimum fall constraint for net or path.</td>
</tr>
<tr>
<td>WIREDLOGIC netName</td>
<td>Not supported.</td>
</tr>
<tr>
<td>MAXDIST distance</td>
<td></td>
</tr>
</tbody>
</table>

BLOCKAGES Section

Blockages give the placement or routing obstruction. No components can be placed and no wiring can be done at locations having blockages.

BLOCKAGES numBlockages

...
Blockages with the given geometry are created as rectangular objects on the layer layerName and purpose boundary. If the layerName is not specified, the rectangle is created on all the masterslice layers defined in the technology file. The property type associated with the rectangular objects is String.

The rectangle object is associated to the instance compName by creating a parent-child relation so that when the instance moves, the blockage also moves along with it. In addition, DEF In creates a group called defBlockages and associates the rectangular object as a member of this group.

To identify the type of obstruction as placement, routing, slots, fills, and pushdown, the properties that will be created on the rectangular object are listed in the table below.

<table>
<thead>
<tr>
<th>DEF 5.4 Statement</th>
<th>LAYER</th>
<th>Property Name</th>
<th>Property Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>- LAYER layerName</td>
<td></td>
<td>defFillsBlockages</td>
<td>DEFAULT</td>
</tr>
<tr>
<td>+ FILLS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DEF Description

| numBlockages      | Specifies the number of blockages in a design. |
| LAYER layerName   | Specifies the routing layer on which to create a blockage. |
| COMPONENT compName| Creates a routing blockage associated with the specified component. If the component is moved, the routing blockage moves with it. |
| PLACEMENT compName| Creates a placement blockage associated with the specified component. If the component is moved, the placement blockage moves with it. You cannot create a floating placement blockage, but you can manually edit the blockage after it is created. |
| RECT pt pt [RECT pt pt] | Specifies the blockage geometry. |
Slots and fills can also cause blockages.

**SLOTS Section**

Slots are physical cuts in a metal layer. They are created as rectangular objects on the layer `layerName` and purpose boundary. Slots are represented as rectangles having lower left and upper right coordinates.

```plaintext
{SLOTS numslots;
   - { LAYER layerName
       RECT pt pt [RECT pt pt] ...;
   }
```

DEF In creates a group called defSlots and associates the rectangular object as a member of this group.

FILLS Section

Fills are metal fills used to fill corresponding slots. They are used to balance the density and other physical characteristics of a metal layer. By default, metal fills are created as rectangular objects on the layer layerName and the boundary purpose. You can use the PURPOSE_FOR_METAL_FILL environment variable to specify a valid purpose on which the metal fill geometries will be created. If you specify an invalid purpose, the metal fill geometries will be created on the boundary purpose, which is the default purpose.

Fills are represented as rectangles having lower left and upper right coordinates.

```plaintext
[FILLS numFills;
  - { LAYER layerName
      [RECT pt pt] ...;
  }
]
```

DEF In creates a group called defFills and associates the rectangular object as a member of this group.

Note: Slots and fills can be created only in the routing layer.
SCANCHAINS Section

Defines scan chains in a design. Scan chains are a collection of cells that contain both scan-in and scan-out pins. These pins must be defined in the PINS section of the DEF file with +USE SCAN.

```
SCANCHAINS numScanChains
   - chainName
      [+ PARTITION partitionName [MAXBITS maxBits]]
      [+ COMMONSCANPINS [ (IN pin) ] [ (OUT pin) ]]
      [+ START {fixedInComp | PIN} {outPin}]
      [+ FLOATING {floatingComp [ (IN pin) ] [ (OUT pin) ]} [BITS numBits]]
      [+ ORDERED {fixedComp [ (IN pin) ] [ (OUT pin)] [BITS numBits]}]
      [+ STOP {fixedOutComp | PIN} {inPin}]
   ;...
END SCANCHAINS
```

The entire SCANCHAINS section is mapped as a property on the cellview with the property SCANCHAINS, property type dbcILLListType, and property value as a list of all the statements.
SKILL Functions

ldtrDefRead

ldtrDefRead(
    t_fileName
    t_libName
    t_cellName
    t_viewName
    [t_refLibs]
    [t_engineName]
    [g_startupNameMap]
    [g_connectivity]
    t_viewNames
)
=>t/nil

Description

Reads a DEF file into a cellview.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_fileName</td>
<td>Input file name</td>
</tr>
<tr>
<td>t_libName</td>
<td>Library name</td>
</tr>
<tr>
<td>t_cellName</td>
<td>Cell name</td>
</tr>
<tr>
<td>t_viewName</td>
<td>View name</td>
</tr>
<tr>
<td>t_refLibs</td>
<td>Reference library list separated by spaces. The default value is &quot;&quot;. Other values can be &quot;mycell1 mycell2&quot;.</td>
</tr>
<tr>
<td>t_engineName</td>
<td>P&amp;R engine name. Values can be SE or GE. The default value is SE.</td>
</tr>
<tr>
<td>g_startupNameMap</td>
<td>Startup name map. The value can be t if the design is retrieving or nil if the design is non retrieving. The default value is nil.</td>
</tr>
<tr>
<td>g_connectivity</td>
<td>Connectivity to preserve or delete. Values can be t or nil. If the value is t, the connectivity in the DEF file and cellview will match. In addition, the instances, nets, and instTerms which are in</td>
</tr>
</tbody>
</table>
cellview but not in DEF will be deleted. If the value is nil, cellview will be updated incrementally from DEF file. The default value is nil.

\textit{t\_viewNames} \hspace{1cm} \text{View name list for the component master. The view names in the list are separated by a spaces. The default value is ‘‘’.}

\begin{description}
\item[t] Returns \textit{t} on the successful execution of the function.
\item[\texttt{nil}] Returns \texttt{nil} when the function encounters an error.
\end{description}

\textbf{Example}

\texttt{ldtrDefRead ("complete.5.2.def" "complete" "muk" "autoLayout")}
ldtrDefWrite

ldtrDefWrite(  
  t_fileName  
  t_libName  
  t_cellName  
  t_viewName  
  [t_engineName]  
  [g_overwriteFile]  
  [t_defLayerName]  
)  
=>t/nil

Description

Writes a cellview to a DEF file.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_fileName</td>
<td>Input file name</td>
</tr>
<tr>
<td>t_libName</td>
<td>Library name</td>
</tr>
<tr>
<td>t_cellName</td>
<td>Cell name</td>
</tr>
<tr>
<td>t_viewName</td>
<td>View name</td>
</tr>
<tr>
<td>t_engineName</td>
<td>P&amp;R engine name. Values can be SE or GE. The default value is SE.</td>
</tr>
<tr>
<td>g_overwriteFile</td>
<td>Enables overwriting an output file. The values is t if the file can be overwritten or nil if the file cannot be overwritten. The default value is t.</td>
</tr>
<tr>
<td>t_defLayerName</td>
<td>Specifies the layer name that is to be exported.</td>
</tr>
</tbody>
</table>

Value Returned

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Returns t on the successful execution of the function.</td>
</tr>
<tr>
<td>nil</td>
<td>Returns nil when the function encounters an error.</td>
</tr>
</tbody>
</table>
Example

ldtrDefWrite("out.def" "complete" "muk" "autoLayout")
ldtrGenLefoutList

ldtrGenLefoutList(
    [g_appendToFile]
    [t_libNamePattern]
    [t_cellNamePattern]
    [t_viewNamePattern]
    [t_fileName]
    [t_cellType]
)
=>t/nil

Description

Generates the lefout.list utility.

Arguments

g_appendToFile  Appends or overwrites the output file. The values is t if it appends to the file or nil if it overwrites the file. The default value is t.

t_libNamePattern  Regular expression capturing library names. The default value is “.*”. Other values can be “^a.*”or “[abc]*$”.

t_cellNamePattern  Regular expression capturing cell names. The default value is “.*”. Other values can be “^a.*”or “[abc]*$”.

t_viewNamePattern  Regular expression capturing view names. The default value is “^abstract$”. Other values can be “.*” or “^layout$”.

t_fileName  Output file name. It can be the name of any file that has a list of cellviews to be written out. The default value is lefout.list.

t_cellType  Types of macros to be written out. The default value is any. Other values can be stdCell, I0Cell, Block, Site, or BaseArray.

Value Returned

t  Returns t on the successful execution of the function.

nil  Returns nil when the function encounters an error.
Example

ldtrGenLefoutList(nil) ; overwrite file lefout.list
ldtrLefRead

ldtrLefRead(
  t_fileName
  t_libName
  [t_refLibs]
  [t_engineName]
  [t_macroView]
  [t_macroabsViewName]
  [t_macrolayViewName]
  [g_connectivity]
  [t_skipDbLock]
)
=>t/nil

Description
Reads a LEF file into a library.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_fileName</td>
<td>Input file name</td>
</tr>
<tr>
<td>t_libName</td>
<td>Library name</td>
</tr>
<tr>
<td>t_refLibs</td>
<td>Reference library list separated by a whitespace. The default value is &quot;&quot;. Other values can be &quot;mycell1 mycell2&quot;</td>
</tr>
<tr>
<td>t_engineName</td>
<td>P&amp;R engine name. Values can be “SE” or “GE”. The default value is “SE”</td>
</tr>
<tr>
<td>t_macroView</td>
<td>View type of the macro. Values can be either “abstract” or “layout”. The default value is “abstract”.</td>
</tr>
<tr>
<td>t_macroabsViewName</td>
<td>View name. This value is taken only if the view type is “abstract”. The default view name is “abstract”.</td>
</tr>
<tr>
<td>t_macrolayViewName</td>
<td>View name. This value is taken only if the view type is “layout”. The default view name is “layout”.</td>
</tr>
<tr>
<td>g_connectivity</td>
<td>Existing connectivity in layout view to be deleted. Values can be t or nil. The default value is nil.</td>
</tr>
</tbody>
</table>
**t_skipDbLock**

Prevents dd object locking during translation. Values can be t or nil. The default value is nil.

**Value Returned**

- **t**
  
  Returns t on the successful execution of the function.

- **nil**
  
  Returns nil when the function encounters an error.

**Example**

```
ldtrLefRead("complete.5.2.lef" "complete")
```
ldtrLefWrite

ldtrLefWrite(
  t_fileName
  t_techlibName
  [g_overwriteFile]
  [l_cvListFileName]
  [g_outputMode]
  [t_engineName]
  [t_lefLayerName]
)
=>t/nil

Description

Writes a library in the LEF format.

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_fileName</td>
<td>Input file name</td>
</tr>
<tr>
<td>t_techlibName</td>
<td>Library name to be written out</td>
</tr>
<tr>
<td>g_overwriteFile</td>
<td>Overwrites an output file. The values is t if the file is overwritten and nil if the file is not overwritten. The default value is nil.</td>
</tr>
<tr>
<td>l_cvListFileName</td>
<td>File consisting of macros used to write a LEF file. It can be the name of any file that has a list of cellviews to be written out. The default value is lefout.list.</td>
</tr>
<tr>
<td>g_outputMode</td>
<td>Retrieves both physical and logical information in the LEF file. The physical information includes information about the cells corresponding to macros, sites, and arrays. The values is t if retrieves both the logical and physical information in a LEF file or nil if it retrieves only the logical information in a LEF file. By default, the value is t.</td>
</tr>
<tr>
<td>t_engineName</td>
<td>P&amp;R engine name. Values can be SE or GE. The default value is SE.</td>
</tr>
<tr>
<td>t_lefLayerName</td>
<td>Specifies a file containing layer names which are to be exported in the output LEF file.</td>
</tr>
</tbody>
</table>
Value Returned

\begin{itemize}
\item \texttt{t} \quad \text{Returns \texttt{t} on the successful execution of the function.}
\item \texttt{nil} \quad \text{Returns \texttt{nil} when the function encounters an error.}
\end{itemize}

Example

\begin{verbatim}
ldtrLefWrite("out.lef" "complete")
\end{verbatim}
Stream Format

This appendix describes the following:

- **Overview** on page 430
- **Stream File Description** on page 431
- **Differences in Stream Format Versions** on page 449
- **Stream Syntax** on page 449
- **Example of a Stream Format File** on page 451
- **How to read, write, and print Stream dates to avoid Y2K problems** on page 460
Overview

When you want to transfer Virtuoso data to other systems, the best format to translate your data to is Stream format. This appendix describes the Stream format used in the files you can produce and read in using the `pipo` utility from a UNIX command line or the Stream In and Stream Out commands from the Translators menu in the CIW.

You can easily transfer libraries preserved in Stream format to other systems for processing. In addition, Stream format is upward compatible, which means newer releases of the Stream translators can read libraries produced with an older release.

For complete descriptions of the options and arguments for Stream In, Stream Out, and `pipo` commands, refer to Chapter 3, “Translating Stream Files”.

Although you can use the Stream In and Stream Out commands to translate your Stream data, you might want to directly edit your Stream file, or write programs or scripts that manipulate your Stream data.

Note: Portions of this appendix describe features and data types that were applicable to some older tools (like GDSII Custom Plus and Construct systems). These features might not apply to the Virtuoso system.

About GDSII

History

The GDSII format was invented at Calma Co. in the early 1970s, which is now a part of Cadence Design Systems, Inc. Therefore, GDSII is today a registered trademark of Cadence.

GDSII Format

GDSII is the open-standard Stream file format for transferring or archiving two-dimensional graphical design data. It is a binary, platform-independent format. You will read more about the Stream format in the subsequent sections. For a sample of the GDSII file, see Example of a Stream Format File on page 451.
Stream File Description

The information stored in a Stream file is coded in variable-length records. The length of a record is measured in eight-bit bytes. The minimum record length is four bytes. There is no limit on record length. Within the record, two bytes (16 bits) is a word. The 16 bits in a word are numbered 0 to 15, left to right.

The first four bytes of a record are the header. The first two bytes of the header specify how many eight-bit bytes the record contains. The third byte of the header specifies the record type. The fourth byte of the header specifies the type of data contained within the record. The fifth through last bytes of the record are data. The next record begins immediately after the last byte in the record.

The following figure shows a typical record header:

<table>
<thead>
<tr>
<th>Bit #</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Record Type</td>
<td>Total record length (in bytes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Data Type</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If the Stream file is on a magnetic tape, the records of the library are usually divided in 2048-byte physical blocks. Records can overlap physical block boundaries; a record is not required to be wholly contained in a single physical block.

Two consecutive zero bytes are a null word. You can use null words to fill the space between the last record of a library and the end of its physical block.

Stream records are always an even number of bytes. If a record contains ASCII string data and the ASCII string is an odd number of bytes, the last character is a null character.

Stream Data Types

The following table lists the Stream data types and their values. The data type is specified by the fourth byte of the record.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>No data present</td>
<td>0</td>
</tr>
<tr>
<td>Bit array</td>
<td>1</td>
</tr>
</tbody>
</table>
The following paragraphs describe these Stream data types. As a reminder, a word consists of 16 bits, numbered 0 to 15, left to right.

- **Bit array (1)**

  A bit array is a word that contains bits, or a group of bits, that represent data. A bit array allows one word to contain more than one piece of information.

- **Two-byte signed integer (2)**

  2-byte integer = 1 word 2s-complement representation

  The range of two-byte signed integers is -32,768 to 32,767.

  The following is a representation of a two-byte integer, where $S$ is the sign and $M$ is the magnitude.

  \[
  S MMM MMM MMM MMM MMM MMM
  \]

  The following are examples of two-byte integers:

<table>
<thead>
<tr>
<th>Value</th>
<th>Two-byte signed integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>00000000 00000001</td>
</tr>
<tr>
<td>3</td>
<td>00000000 00000010</td>
</tr>
<tr>
<td>4</td>
<td>00000000 10001001</td>
</tr>
<tr>
<td>-1</td>
<td>11111111 11111111</td>
</tr>
<tr>
<td>-2</td>
<td>11111111 11111110</td>
</tr>
<tr>
<td>-137</td>
<td>11111111 01110111</td>
</tr>
</tbody>
</table>

- **Four-byte signed integer (3)**

  4-byte integer = 2 word 2s-complement representation

  The range of four-byte signed integers is -2,147,483,648 to 2,147,483,647.

  The following is a representation of a four-byte integer, where $S$ is the sign and $M$ is the magnitude.

  \[
  S MMM MMM MMM MMM MMM MMM MMM MMM
  \]

  The following are examples of four-byte integers:
Four-byte real (4) and eight-byte real (5)

4-byte real = 2-word floating point representation

8-byte real = 4-word floating point representation

For all nonzero values:

A floating point number has three parts: the sign, the exponent, and the mantissa.

- The value of a floating point number is defined as
  (Mantissa) x (16 raised to the true value of exponent field).

- The exponent field (bits 1-7) is in Excess-64 representation. The field shows a number 64 greater than the actual exponent.

- The mantissa is always a positive fraction greater than or equal to 1/16 and less than 1. For a 4-byte real, the mantissa is bits 8 through 31. For an 8-byte real, the mantissa is bits 8 through 63. The binary point is just to the left of bit 8. Bit 8 represents the value 1/2, bit 9 represents 1/4, and so on.

To keep the mantissa in the range of 1/16 to 1, the results of floating point arithmetic are normalized. Normalization is a process that shifts the mantissa left one hex digit at a time until its left FOUR bits represent a non-zero quantity. For every hex digit shifted, the exponent is decreased by one. Since the mantissa is shifted four bits at a time, it is possible for the left three bits of the normalized mantissa to be zero. A zero value, also called true zero, is represented by a number with all bits zero.

The following are representations of 4-byte and 8-byte reals, where \( S \) is the sign, \( E \) is the exponent, and \( M \) is the magnitude. Examples of 4-byte reals are included on the following pages, although 4-byte reals are not used currently. The representation of the negative values of real numbers is exactly the same as the positive, except that the highest order bit is 1, not 0.

In the eight-byte real representation, the first four bytes are exactly the same as in the four-byte real representation. The last four bytes contain additional binary places for higher resolution.

4-byte real:

\[ S E E E E E E E \ M M M M \ M M M M \ M M M M \]

8-byte real:
Examples of 4-byte real:

In the first six lines of the following example, the 7-bit exponent field is 65. The actual exponent is 65-64=1.

01000001 00010000 00000000 00000000 = 1
01000001 00100000 00000000 00000000 = 2
01000001 00110000 00000000 00000000 = 3
11000001 00010000 00000000 00000000 = -1
11000001 00100000 00000000 00000000 = -2
11000001 00110000 00000000 00000000 = -3
01000000 10000000 00000000 00000000 = 0.5
01000000 10011001 10011001 10011001 = 0.6
01000000 10110011 00110011 00110011 = 0.7
01000001 00011000 00000000 00000000 = 1.5
01000001 00011001 10011001 10011001 = 1.6
01000001 00011011 00110011 00110011 = 1.7
00000000 00000000 00000000 00000000 = 0
01000001 00100000 00000000 00000000 = 1
01000010 01100100 00000000 00000000 = 10
01000011 00111110 00000000 00000000 = 100
01000100 00100111 00010000 00000000 = 1000
01000101 00011000 01101010 00000000 = 10000

- ASCII string (6)

A collection of bytes representing ASCII characters. All odd-length strings are padded with a null character (the number zero), and the byte count for the record containing the ASCII string includes this null character. If you write a program to read Stream data, the program must check for the null character and, if present, decrease the length of the string by one.

Stream Records

This section describes the records that make up a Stream file. The descriptions include the following:

- Record number
- Record name
- Data type
- The four digits that make up the second word in the record, divided as follows:
  - The first two digits are the hexadecimal value of the record number.
  - The second two digits are the numeric value for the data type.
A description of the information contained in the record.

0 HEADER Two-Byte Signed Integer.

[0002] Contains the Stream version number.

1 BGNLIB Two-Byte Signed Integer.

[0102] This record type marks the beginning of a library. It contains the last modification time of a library (one word each for years since 1900, month, day, hour, minute, and second), the time of last access (same format). The time fields correspond exactly with the contents of a struct tm, except for the month, which is (tm_mon + 1). See the table for the BGNLIB description in Example of a Stream Format File on page 451, and How to read, write, and print Stream dates to avoid Y2K problems on page 460.

The following figure shows the meaning of each word.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1C (hex) # of bytes in record</td>
<td>01 (hex)</td>
<td>02 (hex)</td>
<td>year (last modification time)</td>
<td>month</td>
<td>day</td>
<td>hour</td>
<td>minute</td>
<td>second</td>
<td>year (last access time)</td>
<td>month</td>
<td>day</td>
<td>hour</td>
<td>minute</td>
</tr>
</tbody>
</table>
2 LIBNAME ASCII String
[0206] Contains the library name. The library name must follow UNIX filename conventions for length and valid characters. The library name can include the file extension (.sf or .db in most cases).

3 UNITS Eight-Byte Real.
[0305] Contains two numbers indicating the number of user units in a database unit and the size of a database unit in meters. Typically, the number of user units in a database unit is less than 1 because you use more than 1 database unit per user unit. To calculate the size of a user unit in meters, divide the second number by the first.

For example, if you create a library using the default unit (user unit = 1 micron and 1000 database units per user unit), the first number is .001 and the second number is 1E−9.

4 ENDLIB No Data Present.
[0400] Marks the end of a library.

5 BGNSTR Two-Byte Signed Integer.
[0502] Contains the creation time and last modification time of a structure (in the same format as the BGNLIB record), and marks the beginning of a structure.

6 STRNAME ASCII String.
[0606] Contains the structure name. The limitation of the size of string in STRNAME record in Stream format was 32 characters. This limit has been removed. Legal structure name characters are

A through Z
a through z
0 through 9
Underscore (_) Question mark (?) Dollar sign ($)}

7 ENDSR No Data Present
[0700] Marks the end of a structure.
8  BOUNDARY  No Data Present
   [0800]  Marks the beginning of a boundary element.
9  PATH  No Data Present
   [0900]  Marks the beginning of a path element.
10  SREF  No Data Present
   [0A00]  Marks the beginning of an SREF (structure reference) element.
11  AREF  No Data Present
   [0B00]  Marks the beginning of an AREF (array reference) element.
12  TEXT  No Data Present
   [0C00]  Marks the beginning of a text element.
13  LAYER  Two-Byte Signed Integer
   [0D02]  Specifies the layer number. The value of the layer must be in the range of 0 to 255.
14  DATATYPE  Two-Byte Signed Integer
   [0E02]  Specifies the data type. The value of the datatype must be in the range of 0 to 255.
15  WIDTH  Four-Byte Signed Integer
   [0F03]  Specifies the width of a path in database units. If the value for the width is negative, the width is absolute and is not affected by the magnification factor of any parent reference. If this record type is omitted, the default is zero.
16  XY  Four-Byte Signed Integer
   [1003]  Contains an array of the XY coordinates for path, boundary, text, contact, SREF, node, box, and AREF elements. The coordinates are in database units. Each X or Y coordinate is four bytes long.
The elements have the following number of coordinates:

- Path elements have a minimum of 2 and a maximum of 8000 coordinates.
- Boundary elements can have a minimum of 4 and a maximum of 8000 coordinates. The first and last coordinates must coincide.
- A text, contact, or SREF element can have only one coordinate. The coordinate for text and SREF is the origin and reference is the origin of the layout of a cell under consideration.
- A node can have from 1 to 50 coordinates.
- A box must have five coordinates, with the first and last coordinates coinciding.
- An AREF has exactly three coordinates. In an AREF, the first coordinate is the array reference point (origin point). The second coordinate locates a position that is displaced from the reference point by the inter-column spacing times the number of columns. The third coordinate locates a position that is displaced from the reference point by the inter-row spacing times the number of rows. The following is an example of an array lattice.

![Array Lattice Diagram]

```
17 ENDEL
[1100] Marks the end of an element.
18 SNAME ASCII String
[1206] Contains the name of a referenced structure. See also STRNAME.
19 COLROW Two-Byte Signed Integer
[1302] Contains four bytes. The first two bytes contain the number of columns in the array. The third and fourth bytes contain the
```
number of rows. The number of columns and the number of rows must be in the range 0 to 32,767 (decimal).

20 TEXTNODE  No Data Present (Not currently used)

[1400] Marks the beginning of a text node.

21 NODE  No Data Present

[1500] Marks the beginning of a node.

22 TEXTTYPE  Two-Byte Signed Integer

[1602] Specifies the text type. The value of the text type must be in the range 0 to 255.

23 PRESENTATION  Bit Array

[1701] Specifies how text is presented.

The bits in the bit array have the following values:

■ Bits 10 and 11, used together as a binary number, specify the font (00 is font 0, 01 is font 1, 10 is font 2, and 11 is font 3).

■ Bits 12 and 13 specify the vertical justification (00 means top, 01 means middle, and 10 means bottom).

■ Bits 14 and 15 specify the horizontal justification (00 means left, 01 means center, and 10 means right).

■ Bits 0 through 9 are reserved for future use and must be cleared.
If this record is omitted, top-left justification and font 0 are the default values. The following shows a PRESENTATION record.

<table>
<thead>
<tr>
<th>Bit #</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6 (hex) # of bytes in record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>17 (hex)</td>
<td>01 (hex)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**24 SPACING** Not currently used

**25 STRING** ASCII String

[1906] Contains up to 512 characters of text to present.

**26 STRANS** Bit Array

[1A01] Contains two bytes of bit flags for SREF, AREF, and text transformation. Bit 0 (the leftmost bit) specifies reflection.

The bits in the bit array have the following values:

- If bit 0 is set, the element is reflected about the X-axis before angular rotation. For an AREF, the entire array is reflected with the individual array members rigidly attached.
- Bit 13 flags absolute magnification.
- Bit 14 flags absolute angle.
- Bits 1 to 12 and 15 are reserved for future use and must be clear.
If this record is omitted, the defaults for the element are no reflection, non-absolute magnification, and non-absolute angle. The following shows an STRANS record.

<table>
<thead>
<tr>
<th>Bit #</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6 (hex) # of bytes in record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1A (hex)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>01 (hex)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unused</td>
</tr>
</tbody>
</table>

27 MAG

Eight-Byte Real

[1B05]
Contains the magnification factor. If this record is omitted, the default magnification factor is one.

28 ANGLE

Eight-Byte Real

[1C05]
Contains the angular rotation factor. The angle of rotation is measured in degrees and in the counterclockwise direction. For an AREF, the ANGLE rotates the entire array (with the individual array members rigidly attached) about the array reference point. If this record is omitted, the default angle is zero degrees.

29 UINTGER

Not currently used. User Integer data was used in GDSII Release 2.0 only.

30 USTRING

Not currently used. User String data, formerly called character string data (CSD), was used in GDSII Releases 1.0 and 2.0.

31 REFILBS

ASCII String

[1FO6]
Contains the names of the reference libraries. This record must be present if any reference libraries are bound to the working library. The name of the first reference library starts at byte 5 (immediately following the record header) and continues for 44 bytes. The next 44 bytes contain the name of the second library. The record is extended by 44 bytes for each additional library (up to 15) which is bound for reference. The reference library names
Design Data Translator's Reference
Stream Format

may include directory specifiers (separated with “/”) and an extension (separated with “.”). If either the first or second library is not named, its place is filled with nulls.

32  FONTS  ASCII String

[2006]  Contains the names of the text font definition files. This record must be present if any of the four fonts have a corresponding text font definition file. This record must not be present if none of the fonts have a text font definition file. The filename for text font 0 starts the record, followed by the filenames for the remaining three fonts. Each filename is 44 bytes long. The filename is padded with nulls if the name is shorter than 44 bytes. The filename is null if no filename corresponds to the font. The filenames may include directory specifiers (separated with “/”) and an extension (separated with “.”).

33  PATHTYPE  Two-Byte Signed Integer

[2102]  Contains a value indicating the type of path endpoints. The value is 0 for square-ended paths that end flush with their endpoints, 1 for round-ended paths, 2 for square-ended paths that extend a half-width beyond their endpoints, and 4 for paths with variable square-ended extensions (see records 48 and 49). If not specified, assumes a type of 0.
The following illustration shows the path types.

<table>
<thead>
<tr>
<th>Path type 0</th>
<th>Path type 0 produces a square-ended path, ending flush with the digitized endpoints. This is the default path type if none is specified.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path type 1</td>
<td>Path type 1 produces a round-ended path. The two ends are semicircular with center at the digitized endpoints.</td>
</tr>
<tr>
<td>Path type 2</td>
<td>Path type 2 produces a square-ended path. The ends of the path extend beyond the digitized endpoints by one-half the path width.</td>
</tr>
<tr>
<td>Path type 4</td>
<td>Path type 4 produces a square-ended path. The ends of the path extend beyond the digitized endpoints or endpoints extend beyond the path by a variable, user-definable distance. The two possibilities are shown here: <img src="image" alt="Path type 4 possibilities" /></td>
</tr>
</tbody>
</table>

### 34 GENERATIONS

Two-Byte Signed Integer

Contains the number of copies of deleted or back-up structures to retain. This number must be at least 2 and not more than 99. If the GENERATIONS record is omitted, the default value is 3.

### 35 ATTRTABLE

ASCII String

Contains the name of the attribute definition file. This record is present only if an attribute definition file is bound to the library. The attribute definition filename can include directory specifiers (separated with “/”) and an extension (separated with “.”). The maximum record size is 44 bytes.
36 STYPTABLE  ASCII String (Unreleased feature)

[2406]

37 STRTYPE  Two-Byte Signed Integer (Unreleased feature)

[2502]

38 ELFLAGS  Bit Array

[2601]  Contains two bytes of bit flags. Bit 15 (the rightmost bit) specifies Template data. Bit 14 specifies External data (also referred to as Exterior data). All other bits are currently unused and must be cleared to 0. If this record is omitted, the default value for all bits is 0.

The following illustration shows an ELFLAGS record.

```
<table>
<thead>
<tr>
<th>Bit #</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6 (hex) # of bytes in record</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 2</td>
<td></td>
<td>26 (hex)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>External data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Word 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unused</td>
</tr>
<tr>
<td></td>
<td>Template data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

39 ELKEY  Four-Byte Signed Integer (Unreleased feature)

[2703]

40 LINKTYPE  Two-Byte Signed Integer (Unreleased feature)

[28]

41 LINKKEYS  Four-Byte Signed Integer (Unreleased feature)

[29]

42 NODETYPE  Two-Byte Signed Integer
Contains a value indicating the node type. The value of the node type must be in the range of 0 to 255.

Two-Byte Signed Integer

Contains the attribute number. The attribute number is an integer from 1 to 127. Attribute numbers 126 and 127 are reserved for the user integer and user string (CSD) properties which were used prior to Release 3.0.

ASCII String

Contains the string value associated with the attribute named in the preceding PROPATTR record. The maximum string length is 126 characters. The attribute-value pairs associated with any one element must all have distinct attribute numbers. Also, the total amount of property data that can be associated with any one element is limited: the total length of all the strings, plus twice the number of attribute-value pairs, must not exceed 128. (Or, if the element is an SREF, AREF, contact, nodeport, or node, the length must not exceed 512.) For example, if a boundary element uses property attribute 2 with property value “metal,” and property attribute 10 with property value “property,” the total amount of property data is 18 bytes. This is 6 bytes for “metal” (odd-length strings must be padded with a null) plus 8 for “property” plus 2 times the 2 attributes (4), which equals 18.

No Data Present

Marks the beginning of a box element.

Two-Byte Signed Integer

Contains a value indicating the box type. The value of the box type must be in the range of 0 to 255.

Four-Byte Signed Integer

A unique positive number that is common to all elements of the plex to which this element belongs. The head of the plex is flagged by setting the seventh bit; therefore, plex numbers
should be small enough to occupy only the rightmost 24 bits. If this record is not present, the element is not a plex member.

48 BGNEXTN Four-Byte Signed Integer

[3003] Applies to Pathtype 4. Contains four bytes that specify in database units the distance a path outline begins before or after the first point of the path. This value can be negative.

49 ENDEXTN Four-Byte Signed Integer

[3103] Applies to Pathtype 4. Contains four bytes that specify in database units the distance a path outline begins before or after the last point of the path. This value can be negative.

50 TAPENUM Two-Byte Signed Integer

[3202] Contains the number of the current reel of tape for a multi-reel Stream file. For the first tape, the tape number is 1; for the second tape, the tape number is 2. For each additional tape, increment the tape number by one.

51 TAPECODE Two-Byte Signed Integer

[3302] Contains a unique, 6-integer code common to all reels of a multi-reel Stream file. It verifies that the correct reels are being read.

52 STRCLASS Two-Byte Bit Array (Cadence internal use only)

[3401] If Stream tapes are produced by non-Cadence programs, this record should either be omitted or cleared to zero.

53 RESERVED Not currently used

[3503] This record type was used for NUMTYPES but was not required.

54 FORMAT Two-Byte Signed Integer

[3602] This optional record defines the format of a Stream tape in two bytes. The possible values are

0 for GDSII Archive format
1 for GDSII Filtered format
2 for EDSIII Archive format
3 for EDSIII Filtered format

An Archive Stream file contains elements for all the layers and data types. In an Archive Stream file, the FORMAT record is followed immediately by the UNITS record. A file that does not have the FORMAT record is assumed to be an Archive file.

A Filtered Stream file contains only the elements on the layers and with the datatypes you specify during creation of the Stream file. The list of layers and datatypes specified appear in MASK records. At least one MASK record must immediately follow the FORMAT record. The MASK records are terminated with the ENDMASKS record.

55 MASK
ASCII String

This record is required for and present only in Filtered Stream files. It contains the list of layers and data types specified when the file was created. At least one MASK record must immediately follow the FORMAT record. More than one MASK record can occur. The last MASK record is followed by the ENDMASKS record.

In the MASK list, datatypes are separated from the layers with a semicolon. Individual layers or datatypes are separated with a space. A range of layers or datatypes is specified with a dash. An example MASK list looks like this:

    1 5-7 10 ; 0-255

56 ENDMASKS
No Data Present

This record is required for and present only in Filtered Stream files. It marks the end of the MASK records. The ENDMASKS record must follow the last MASK record. ENDMASKS is immediately followed by the UNITS record.

57 LIBDIRSIZE
Two-Byte Signed Integer

Contains the number of pages in the library directory.

58 SRFNAME
ASCII String
## Stream Format

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[3A06]</td>
<td>Contains the name of the spacing rules file, if one is bound to the library.</td>
</tr>
<tr>
<td>59</td>
<td>LIBSECUR Two-Byte Signed Integer</td>
</tr>
<tr>
<td>[3B02]</td>
<td>Contains an array of Access Control List (ACL) data. Each ACL entry consists of a group number, a user number, and access rights. Up to 32 ACL entries can be present.</td>
</tr>
<tr>
<td>60</td>
<td>BORDER No Data Present</td>
</tr>
<tr>
<td>[3C00]</td>
<td>Marks the beginning of a border element.</td>
</tr>
<tr>
<td>61</td>
<td>SOFTFENCE No Data Present</td>
</tr>
<tr>
<td>[3D00]</td>
<td>Marks the beginning of a soft fence element.</td>
</tr>
<tr>
<td>62</td>
<td>HARDFENCE No Data Present</td>
</tr>
<tr>
<td>[3E00]</td>
<td>Marks the beginning of a hard fence element.</td>
</tr>
<tr>
<td>63</td>
<td>SOFTWIRE No Data Present</td>
</tr>
<tr>
<td>[3F00]</td>
<td>Marks the beginning of a soft wire element.</td>
</tr>
<tr>
<td>64</td>
<td>HARDWIRE No Data Present</td>
</tr>
<tr>
<td>[4000]</td>
<td>Marks the beginning of a hard wire element.</td>
</tr>
<tr>
<td>65</td>
<td>PATHPORT No Data Present</td>
</tr>
<tr>
<td>[4100]</td>
<td>Marks the beginning of a path port element.</td>
</tr>
<tr>
<td>66</td>
<td>NODEPORT No Data Present</td>
</tr>
<tr>
<td>[4200]</td>
<td>Marks the beginning of a node port element.</td>
</tr>
<tr>
<td>67</td>
<td>USERCONSTRAINT No Data Present</td>
</tr>
<tr>
<td>[4300]</td>
<td>Marks the beginning of a user constraint.</td>
</tr>
<tr>
<td>68</td>
<td>SPACER ERROR No Data Present</td>
</tr>
</tbody>
</table>
Differences in Stream Format Versions

There are five available versions of stream format, version 3, 4, 5, 6 and 7. Stream versions 4 and 5 are the same. The records listed in the above section are a combined list of all the records supported by the various stream formats. The various versions provide support for a subset of records present in this list.

The Stream format is usually broken down into 4 versions:

- Stream format version 3 supports basic geometric information. From the list of supported stream records, Stream 3.0 only supports up to record 44. See Stream Records on page 434.

- Stream format version 4 or 5 provides support for geometric information along with supporting new types in GDSII 4, e.g. nodes, ports, template, plex data. From the list of supported stream records, version 5.2 supports up to record 56.

- Stream format 6, has some additional information for a product called "Custom Plus" that includes things like "attach file" and "lib access control lists". From the list of supported stream records, version 6.0 supports up to record 59.

- The last format version is generically called 7 which has lots of the limits removed. In this version, polygons can have more than 200 points, more than 64 layers and datatypes are supported, and more than 10 levels of hierarchy are supported. From the list of supported stream records, version 7 supports all the records.

Stream 6 was very GDSII-specific, so not many non-Calma reader/writer applications handle it. Simple applications deal with Stream 3 (basic geometries), more advanced deal with 4/5 and general purpose remove limitations and can handle up to version 7.

Stream Syntax

This section contains a Bachus Naur representation of the Stream syntax. Bachus Naur uses ALL CAPS to represent the name of an actual record type and lower case for names that can...
be further broken down into a set of actual record types. The following table provides descriptions of the Bachus Naur symbols.

<table>
<thead>
<tr>
<th>Name of Symbol</th>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double colon</td>
<td>::</td>
<td>“Is composed of”</td>
</tr>
<tr>
<td>Square brackets</td>
<td>[ ]</td>
<td>An element that can be absent or occur one time.</td>
</tr>
<tr>
<td>Braces</td>
<td>{}</td>
<td>One of the elements within the braces can occur.</td>
</tr>
<tr>
<td>Braces with an asterisk</td>
<td>{}*</td>
<td>The elements within the braces can be absent or occur one or more times.</td>
</tr>
<tr>
<td>Braces with a plus</td>
<td>{}+</td>
<td>The elements within braces must occur one or more times.</td>
</tr>
<tr>
<td>Angle brackets</td>
<td>&lt;&gt;</td>
<td>These elements are further defined in the Stream syntax list.</td>
</tr>
<tr>
<td>Vertical bar</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<stream format>::=
  HEADER BGNLIB [LIBDIRSIZE] [SRFNAME] [LIBSECUR] LIBNAME [REFLIBS] [FONTS]
  [ATTRTABLE] [GENERATIONS] [<FormatType>] UNITS (..<structure>)* ENDLIB

<FormatType>::= FORMAT | FORMAT {MASK}+ ENDMASKS

<structure>::= BGNSTR STRNAME [STRCLASS] (..<element>)* ENDSTR

<element>::= (..<boundary> | <path> | <SREF> | <AREF>
  | <text> | <node> | <box> ) (..<property>)* ENDEL

<boundary>::= BOUNDARY [ELFLAGS] [PLEX] LAYER DATATYPE XY

<path>::= PATH [ELFLAGS] [PLEX] LAYER DATATYPE [PATHTYPE] [WIDTH] [BGNEXTN]
  [ENDEXTN] XY

<SREF>::= SREF [ELFLAGS] [PLEX] SNAME [..<strans>] XY

<AREF>::= AREF [ELFLAGS] [PLEX] SNAME [..<strans>] COLROW XY

<text>::= TEXT [ELFLAGS] [PLEX] LAYER <textbody>

<node>::= NODE [ELFLAGS] [PLEX] LAYER NODETYPE XY

<box>::= BOX [ELFLAGS] [PLEX] LAYER BOXTYPE XY

<textbody>::=
  TEXTYPE [PRESENTATION] [PATHTYPE] [WIDTH] [..<strans>] XY STRING

<strans>::= STRANS [MAG] [ANGLE]

<property> ::= PROPATTR PROPVALUE
Note: The WIDTH record (inside the TEXT) is not used by the Virtuoso system. The one within PATH is used.

Example of a Stream Format File

The following is an example of a Stream format file. An explanation follows the example.

% od -h example.sf
000 0006 0002 0258 001C 0102 0067 0009 0003 000D 0010 ........X....
008 0000 0000 0000 0067 0009 0003 000D 0010 .......X........
010 0000 0006 3902 0028 000A 3B02 0003 0005 ......9..(..;....
018 0007 0010 0206 6578 616D 706C 652E 6368 ......example.ch
020 7000 005C 1F06 7265 6631 2E63 6870 0000 p.....ref1.chp..
028 0000 0000 0000 0000 0000 0000 0000 0000 ................
030 0000 0000 0000 0000 0000 0000 0000 0000 ................
****
048 0000 0000 0000 0000 0000 0000 0000 00B4 ................
050 2006 6361 6C6D 6166 6F6E 742E 666E 7400  .calmafont.fnt.
058 0000 0000 0000 0000 0000 0000 0000 0000 ................
060 0000 0000 0000 0000 0000 0000 0000 0074 ..............te
068 7874 2E66 6E74 0000 0000 0000 0000 0000 xt.fnt..........
070 0000 0000 0000 0000 0000 0000 0000 0000 ................
078 0000 0000 0000 0000 0000 0000 0000 0000 ......font.f
080 6E74 0000 0000 0000 0000 0000 0000 0000 nt............
088 0000 0000 0000 0000 0000 0000 0000 0000 0000 ........
090 0000 0000 0000 7067 666F 6E74 2E66 6E74 ......pgfont.fnt
098 0000 0000 0000 0000 0000 0000 0000 0000 ................
0A0 0000 0000 0000 0000 0000 0000 0000 0000 ................
0A8 0000 0000 0000 0000 0000 0000 0000 0000 ................
0B0 2220 0003 0014 0305 3E41 8937 4BC6 A7EF ".......>A.7K...
0B8 3944 B2F A09B 5A5D 001C 0502 0067 0007 9D./..ZQ......
0C0 000C 0011 001D 000A 0067 0007 0011 0011 ........X......
0C8 0003 0014 000C 0606 6578 616D 706C 6532 |........example2
0D0 0004 0B00 000C 1206 6578 616D 706C 6531 ......example1
0E0 0008 1302 0002 0002 01C 1003 0000 ..............
0E8 4E20 0000 4E20 0000 4E20 0001 4FF0 0001 N ...N .....0...
0F0 3880 0000 4E20 0004 1100 0004 0700 01C 8..N ........
0F8 0502 0067 0007 000C 0000 0009 0067 ......X...........X
100 0008 001C 000F 0039 003A 000C 0606 6578 ......9.|......ex
108 616D 706C 0004 0C00 0006 0D02 0000 ample1...........
110 0006 1602 0000 0006 1701 0005 0006 1A01 .......
118 8006 000C 1B05 4120 0000 0000 0000 000C ......A .......
120 1003 0000 4E20 0000 4E20 0000 1906 4920 ....N .N ....I.
128 414D 2048 4552 450D 0004 1100 0004 0800 AM HERE........
130 0006 2601 0001 0006 0D02 0002 0006 0E02 ..&.............
138 0003 024A 1003 0000 1388 0000 6D60 0000 ..$ ....m'
140 2EE0 0000 6D60 0000 1F40 0000 84D0 0000 ..m'...m'
148 1388 0000 6D60 0000 1100 0004 0900 0006 ..m'
150 0D02 0004 0006 0E02 003F 0006 2102 0001 .......
158 0008 0F03 0000 0388 0024 1003 0000 3A98 ..$ ....|...
160 0000 36B0 0000 6590 0000 36B0 0000 84D0 .6...e...6...
168 0000 2328 0000 55F0 0000 1770 0006 2B02 ....#|U...p..+
170 0002 000A 2C06 4D45 4552 0006 2B02 .......METAL...
178 000A 000C 2C06 5052 4F50 4552 5459 0004 ....,.PROPERTY.
180 1100 0004 0700 0004 0400 .................
The database that produced this Stream format output has only two structures. They are called example1 and example2. Example1 contains a boundary that is template data, a path with two properties, and a middle-center justified text element containing the string, I AM HERE. Example2 contains only one element, a 2 by 2 AREF of example1.

The following are explanations of the records contained in the example Stream file. As a reminder, the first two words (four bytes) of a record are the record header. The first word shows the record length in bytes, and the second word identifies the record type and the data type.

0006 0002 0258

The first word reports that this record is 6 bytes long. The second word indicates that this is the HEADER (00 hex) record and that the data type is a two-byte signed integer (02 hex). The information in the third word is the Stream version number, which is version 600 (258 hex).

001C 0102 0067 0009 0000 0000 0000 0000 0067 0009 0003 000D 0010 0000

This record is 28 (1C hex) bytes. It is the BGNLIB (01 hex) record. The data type is a two-byte signed integer (02). The remaining 24 bytes of information contain the date and time the library was last modified and the date and time of last access. For example, the last six words of information contain:

<table>
<thead>
<tr>
<th>Time Component</th>
<th>Hexadecimal Representation</th>
<th>Decimal Representation</th>
<th>Real Date Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>year since 1900</td>
<td>0067</td>
<td>00103</td>
<td>2003</td>
</tr>
<tr>
<td>month</td>
<td>0009</td>
<td>00009</td>
<td>9 (September)</td>
</tr>
<tr>
<td>day</td>
<td>0003</td>
<td>00003</td>
<td>3</td>
</tr>
<tr>
<td>hour</td>
<td>000D</td>
<td>00013</td>
<td>1 pm.</td>
</tr>
<tr>
<td>minute</td>
<td>0010</td>
<td>00016</td>
<td>16</td>
</tr>
<tr>
<td>second</td>
<td>0000</td>
<td>00000</td>
<td>0</td>
</tr>
</tbody>
</table>

This record indicates that this library was last accessed on September 3, 2003, at 1:16:00 p.m. Note that this is local time, and that no time zone or daylight savings time information is stored.

0006 3902 0028
This record is 6 bytes. It is the LIBDIRSIZE (39 hex) record. The data type is a two-byte
signed integer (02). In this example, the directory size is 40 (28 hex) pages.

000A 3B02 0003 0005 0007

This record is 10 (A hex) bytes. It is a LIBSECUR (3B hex) record. The data type is a two-byte
signed integer (02). This example has only one ACL entry. The entry has a group number of
3, a user number of 5, and access rights of 7. This means that the only one with any access
rights to this library is user number 5 in group number 3. The access code (007) means this
user has read and write access and is also the owner of the library.

0010 0206 6578 616D 706C 652E 6368 7000

This record is 16 (10 hex) bytes. It is the LIBNAME (02 hex) record. The data type is an ASCII
string (06). The six words of information contain the library name, example.chp.

005C 1F06 7265 6631 2E63 6870 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

This record is 92 (5C hex) bytes. It is the REFLIB (1F hex) record. The data type is an ASCII
string (06). In this example, the library refl.chp is the bound reference library. The library
is padded with nulls to equal 44 bytes. At least 92 bytes of this record must be present if any
reference libraries are bound to the working library. No other reference library is bound, so
the last 44 bytes are filled with nulls. If more than two reference libraries are bound, the record
is extended by 44 bytes for each additional library.

00B4 2006 6361 6C6D 6166 6F6E 742E 666E 7400 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

This record is 180 (B4 hex) bytes. It is the FONTS (20 hex) record. The data type is an ASCII
string (06). All 180 bytes of this record must be present if any textfont files are bound to this
library. In this example, four textfont files (the maximum possible) are bound to the library. The
files are calmafont.fnt, text.fnt, font.fnt, and pgfont.fnt. Each string is
padded with nulls out to 44 bytes.
This record is 12 (C Hex) bytes. It is the ATTRTABLE (23 hex) record. The data type is an ASCII string (06). This record is only present if an attribute table is bound to the library. The name of the attribute table is attrs.at.

This record is 6 bytes. It is the GENERATIONS (22 hex) record. The data type is a two-byte signed integer (02). In this example, three generations of a structure are retained in the library.

This record is 20 (14 hex) bytes. It is the UNITS (03 hex) record. The data type is an eight-byte real (05). In this example, 3E41 8937 4BC6 A7EF is 1E-3. This implies that a database unit is .001 of a user unit. The record 3944 B82F A09B 5A51 is 1E-9. This implies that a database unit is 1E-9 meters (1E-3 microns).

This record is 28 (1C hex) bytes. It is the BGNSTR (05 hex) record. The data type is a two-byte signed integer (02). The information in this record is the creation time and last modification time of the structure. The information is in the same format as the BGNLIB record. This structure was created July 12, 2003, at 5:29:10 p.m. and last modified July 17, 2003, at 5:58:20 p.m.

This record is 12 (C hex) bytes. It is the STRNAME (06 hex) record. The data type is an ASCII string (06). The structure name is example2.

This record is 4 bytes. It is the AREF (0B hex) record. It contains no data (00). It marks the start of an AREF.
This record is 12 (C hex) bytes. It is the SNAME (12 hex) record. The data type is an ASCII string (06). This record contains the name of referenced structure example1.

0006 1A01 8000

This record is 6 bytes. It is the STRANS (1A hex) record. The data type is a bit array (01). In this example, only bit 0 is set, which implies that this AREF is reflected. Since bit 13 and 14 are not set, this structure’s magnification and angle, respectively, are not absolute.

000C 1C05 425A 0000 0000 0000

This record is 12 (C hex) bytes. It is the ANGLE (1C hex) record. The data type is eight-byte real data (05). The data 425A 0000 0000 0000 represents 90.0, which implies that this AREF is placed at an angle of 90 degrees.

0008 1302 0002 0002

This record is 8 bytes. It is the COLROW (13 hex) record. The data type is a two-byte signed integer (02). This example contains a 2 x 2 AREF.

001C 1003 0000 4E20 0000 4E20 0000 4E20 0001 4FF0
0001 3880 0000 4E20

This record is 28 (1C hex) bytes. It is the XY (10 hex) record. The data type is a four-byte signed integer (03). The data, taken two words at a time, can be translated to decimal as 20000, 20000, 20000, 86000, 80000, 20000. Multiply these by .001 (because a data base unit is .001 of a user unit). The results are the coordinates (20, 20), (20, 86), and (80, 20). The first coordinate is the array reference point. The second coordinate is a point that is displaced from the array reference point in the Y-direction by the number of columns times the inter-column spacing. In this example, the second point was displaced 66 (86 - 20) units from the array reference point. Since the array has two columns, this implies that the inter-column spacing is 33 units. A similar calculation can be carried out to verify that the inter-row spacing is 30 units.

0004 1100

This record is 4 bytes. It is the ENDEL (11 hex) record. It contains no data (00). ENDEL marks the end of an element.
0004 0700
This record is 4 bytes. It is the ENDSTR (07 hex) record. It contains no data (00). ENDSTR marks the end of a structure.

001C 0502 0067 0007 000C 000B 001C 0009 0067 0008 001C 000F 0039 003A
This is another BGNSTR record. This structure was created July 12, 2003, at 11:28:09 a.m., and last modified August 28, 2003, at 3:57:58 p.m.

000C 0606 6578 616D 706C 6531
This is another STRNAME record. It contains the string example1.

0004 0C00
This record is 4 bytes. It is the TEXT (0C hex) record. It contains no data (00). Text marks the start of a text element.

0006 0D02 0000
This record is 6 bytes. It is the LAYER (0D hex) record. The data type is a two-byte signed integer (02). This text element is on layer 0.

0006 1602 0000
This record is 6 bytes. It is the TEXTTYPE (16 hex) record. The data type is a two-byte signed integer (02). This text element is texttype 0.

0006 1701 0005
This record is 6 bytes. It is the PRESENTATION (17 hex) record. The data type is a bit array (01). The hex number 0005 in binary has all bits set to zero except bits 13 and 15. Since bits 10 and 11 are 00, the text element is font 0. Since bits 12 and 13 are 01, the text has a middle vertical position. Since bits 14 and 15 are 01, the text has a center horizontal presentation.
This is another STRANS record. This text is reflected and has an absolute magnification and absolute angle.

```
000C 1B05 4120 0000 0000 0000
```

This record is 12 (C hex) bytes. It is the MAG (1B hex) record. The data type is eight-byte real (05). The data in this record represents 2.0, meaning that this text is magnified two times.

```
000C 1003 0000 4E20 0000 4E20
```

This is another XY record. The text is placed at coordinate (20, 20).

```
000E 1906 4920 414D 2048 4552 450D
```

This record is 14 (E hex) bytes. It is the STRING (19 hex) record. The data type is an ASCII string (06). The text string is I AM HERE.

```
0004 1100
```

This is another ENDEL record.

```
0004 0800
```

This record is 4 bytes. It is the BOUNDARY (08 hex) record. It contains no data (00). BOUNDARY marks the start of a boundary element.

```
0006 2601 0001
```

This record is 6 bytes. It is the ELFLAGS (26 hex) record. The data type is a bit array (01). Since bit 15 is set, this element is template data. Since bit 14 is not set, the element is not external data.

```
0006 0D02 0002
```

This is another LAYER record. The boundary is on layer 2.
This record is 6 bytes. It is the DATATYPE (0E hex) record. The data type is a two-byte signed integer (02). This boundary is of datatype 3.

This is another XY record. The coordinates are (5, 28), (12, 28), (8, 34), and (5, 28).

This is another ENDEL record.

This record is 4 bytes. It is the PATH (09 hex) record. It contains no data (00). PATH marks the start of a path element.

This is another LAYER record. The path is on layer 4.

This is another DATATYPE record. The path is datatype 63 (3F hex).

This record is 6 bytes. It is the PATHTYPE (21 hex) record. The data type is a two-byte signed integer (02). This path is pathtype 1.

This record is 8 bytes. It is the WIDTH (0F hex) record. The data type is a four-byte signed integer (03). The number 03E8 hex is 1000 in decimal. Multiply this by .001 (because a database unit is .001 of a user unit). The result is 1; therefore, the width of this path is 1.
This is another XY record. This path's coordinates are (15, 14), (26, 14), (34, 9), and (22, 6).

This record is 6 bytes. It is the PROPATTR (2B hex) record. The data type is a two-byte signed integer (02). This path has a property with attribute number 2.

This record is 10 (A hex) bytes. It is the PROPVALUE (2C hex) record. The data type is an ASCII string (06). The property value for property attribute 2 (above) is METAL. The odd length string (five characters) is padded with a null.

This is another PROPATTR record. This path has another property associated with it. The property has attribute number 10 (A hex).

This is another PROPVALUE record. Property attribute 10 (above) has the value PROPERTY.

This is another ENDEL record.

This is another ENDSTR record.

This record is 4 bytes. This record is the ENDLIB (04 hex) record. It contains no data (00). ENDLIB marks the end of a Stream format file.
How to read, write, and print Stream dates to avoid Y2K problems

The year fields in Stream dates represent the number of years since 1900, and this has led to some misunderstandings, because dates before 2000 appeared to be abbreviated rather than offset by 1900. For example, the year 1999 is represented by decimal 00099 in Stream. This can lead to errors after 2000 if the Stream year fields are assumed to be only tens and units, such as in:

```c
printf("Modification date: %d/%d/19%d\n", stream[2], stream[1], stream[0]);
```

The above code will print “Modification date: 1/1/19103” in the year 2003.

The format of the year fields in Stream corresponds exactly with the year field in a struct tm, and this structure is a very convenient way of getting the necessary 6 integers to store in a Stream date. The month is the only field which needs to be modified, and it needs to be incremented when writing, and decremented when reading Stream. It is recommended that the methods used in the following example be used when writing and reading Stream, and that the date not be printed directly from the Stream contents.

Code Examples

The following code examples are the recommended methods for writing dates to a Stream file, reading and printing dates from a Stream file, and comparing Stream dates. In each case, the date is encoded and decoded to/from a time_t time stamp.

```c
int strmTime[6]; /* buffer of integers read from or to be written to Stream file */
int lowYears, highYears; /* counters for bad year content */

/**************************************************************************
* Encode a time stamp into a time buffer to write to a Stream file. 
**************************************************************************/

void timeStampToStreamBuffer(int *strmTime, time_t timeStamp)
{
    struct tm *ptm;

    ptm = localtime (&timeStamp);
    strmTime[0]= ptm->tm_year; /* Make month 1-based */
    strmTime[1]= ptm->tm_mon + 1;
    strmTime[3]= ptm->tm_hour;
    strmTime[4]= ptm->tm_min;
}
```
```c
static time_t convStrmDateToTime_t(strmTime)
{
    struct tm ptm;
    time_t ttime;

    if (strmTime[0] <= 69) {
        numLowYears++;
        ptm.tm_year = strmTime[0] + 100;
    } else if (strmTime[0] >= 1970) {
        numHighYears++;
        ptm.tm_year = strmTime[0] - 1900;
    } else {
        ptm.tm_year = strmTime[0];
    }

    ptm.tm_mon = strmTime[1] - 1;
    ptm.tm_mday = strmTime[2];
    ptm.tm_hour = strmTime[3];
    ptm.tm_min = strmTime[4];
    ptm.tm_sec = strmTime[5];
    /* initialize these values for mktime. */
    ptm.tm_wday = 0;
    ptm.tm_yday = 0;
    ptm.tm_isdst = -1;
    ttime = mktime(&ptm);
    return(ttime);
}

/****************************
* Example use of the above functions.
* The following code gets the current system time in a time_t, prints
* the date from that time_t, encodes it in an integer buffer, which
* could be written to a Stream file, prints the contents of that
* buffer, decodes the buffer back into a time_t, prints the date from
* that time_t, and then compares the before and after time_t.s.
* Example use of the above functions.
* The following code gets the current system time in a time_t, prints
* the date from that time_t, encodes it in an integer buffer, which
* could be written to a Stream file, prints the contents of that
* buffer, decodes the buffer back into a time_t, prints the date from
* that time_t, and then compares the before and after time_t.s.
*******************************************/
void warnBadDates()
{
    if (lowYears > 0) {
        printf("Warning: years < 70 encountered in Stream file
    printf("Warning: years < 70 encountered in Stream file
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
    printf("Warning: Stream year fields should be number of years
```
main(int argc, char **argv)
{
    time_t timeStamp, newTimeStamp;
    short strmTime[6];
    char timeBuf[80];
    lowYears = highYears = 0;
    time(&timeStamp);
    printf("Input time is %s
", ctime(&timeStamp));
    timeStampToStreamBuffer(strmTime, timeStamp);
    printf("Contents of Stream time buffer: %0.4x %0.4x %0.4x %0.4x
%0.4x %0.4x
", 
        strmTime[0], strmTime[1], strmTime[2],
        strmTime[3], strmTime[4], strmTime[5]);
    newTimeStamp = convStrmDateToTime_t(strmTime);
    printf("Time returned from time buffer: %s
", 
        ctime(&newTimeStamp));
    if (timeStamp != newTimeStamp) {
        printf("Error: timestamps differ: %d %d
", timeStamp, 
                newTimeStamp);
        printf("old: %s
", ctime(&timeStamp));
        printf("new: %s
", ctime(&newTimeStamp));
    } else
        printf("Timestamps matched
");
    warnBadDates();
}
PIPO Messages

Introduction

The translators display messages of different severity levels when specific conditions exist. The severity of message can vary from fatal, error, warning, or informational. The message types are described below:

<table>
<thead>
<tr>
<th>Severity Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fatal</td>
<td>Fatal messages are displayed for high severity issues which force the translator to stop execution.</td>
</tr>
<tr>
<td>Error</td>
<td>Error messages are displayed for high severity issues which might have impact on the accuracy and integrity of the output. Such issues however do not stop the tool from finishing the execution.</td>
</tr>
<tr>
<td>Warning</td>
<td>Warnings are displayed for those issues which might have an impact on the accuracy and integrity of the output but are at the same time not very critical. Such issues do not stop translation.</td>
</tr>
<tr>
<td>Informational</td>
<td>Informational messages are displayed to provide the status of a running process.</td>
</tr>
</tbody>
</table>

This appendix describes various PIPO messages. When generated by PIPO, these messages appear as follows:

`messageType(messageID):message`

where

- `messageType` Severity level of the message.
- `messageID` Message ID of the message.
- `Message` Text of the message displayed.
Note: Words in italics indicate arguments for which you must substitute a name or a value.

Example:

WARNING (19): PIPO does not support attaching text to the object type objectType.

In this example, messageType is WARNING, messageId is 19 followed by the text of the message.

Note: Some PIPO messages have both the FATAL and ERROR severity levels whereas some messages are both FATAL and WARNING. Therefore to avoid the repetition of the messages, the format for describing these messages has been changed as shown below:

Message ID: messageId

Message Type: messageType

Message: message

Extended Help: Provides additional information related to a message.

Example:

Message ID: 1

Message Type: FATAL/WARNING

Message:

Failed to open the library libraryName. Check that library is present in the path defined in the cds.lib file.
Invoking Extended Help

If you require additional information for a particular message, type `msgHelp` at the command prompt followed by the product name and the message number. The syntax for using the `msgHelp` command is described below:

```
msgHelp prod_Name errorNumber1 errorNumber2.
```

where,

- `prod_Name` Product name
- `errorNumber` Error number

Example:

1. Consider the following warning message displayed by PIPO:
   
   WARNING (9): The pcell variant name crossed the limit of 1024 while appending a parameter value.

2. To display extended help for the above warning message, type the following at the command prompt:
   
   `msgHelp PIPO 9`

3. The extended help is displayed as described below:
   
   EXTENDED HELP FOR NUMBER: 9

   If you choose `Params` value for the `Pcell Suffix` option, then Stream Out will create the names of pcell submasters by joining supermaster name and values of different parameters. Only that many parameters will be appended so as to keep the variant name length within the limit of 1024.
Error Messages and Warnings

Message ID: 1
Message Type: FATAL/WARNING
Message:
Failed to open the library libraryName. Check that library is present in the path defined in the cds.lib file.

Message ID: 2
Message Type: FATAL
Message:
Ignoring the Skip Undefined Layer-Purpose Pair option because a new library is being created.

Message ID: 3
Message Type: WARNING
Message:
The Do Not Overwrite Existing Cell and the Keep Stream Cells options have been used together. The Do Not Overwrite Existing Cell option will override the Keep Stream Cells option for cells already existing in the target library.

Message ID: 4
Message Type: FATAL
Message:
Failed to open the technology file technologyFileName from library libraryName. Check that the library exists at the location defined in the cds.lib file and that you have read permission to the technology file.
Message ID: 5
Message Type: WARNING
Message:
Failed to create the pipo_xout_info file in the run directory. Check that you have permission to create this file in the run directory.

Extended Help: The pipo_xout_info file is created for generating design information in the format.libraryName:cellName:viewName:versionList

Message ID: 6
Message Type: FATAL
Message:
Failed to open the primary cellview libraryName/cellName/viewName. Check that the source library exists in the location defined in the cds.lib file, and primary cell exists.

Message ID: 7
Message Type: WARNING
Message:
The value DbIdPlusTime of the Pcell Suffix option might lead to increase in Stream file size.

Extended Help: When you specify DbIdPlusTime as the value for the Pcell Suffix option, Stream Out creates unique names of pcell submasters by joining pcell supermaster name, dbld of submaster, and the time of creation of a cellview. As a result Stream file size increases.

Message ID: 8
Message Type: WARNING
Message:
The parameter parameterName of the pcell pcellName will not be used to form variant name.
Extended Help: When you specify Params value for the Pcell Suffix option, Stream Out creates the names of pcell submasters by joining the pcell supermaster name, and values of different parameters. A parameter will not be used to form the name of a variant if it is not of type int, string, float, boolean or list.

Message ID: 9
Message Type: WARNING
Message:
The pcell variant name crossed the limit of 1024 while appending a parameter value.

Extended Help: When you specify the value Params for the Pcell Suffix option, then Stream Out creates the names of pcell submasters by joining supermaster name, and values of different parameters. Only those parameters will be appended so that the variant name length does not exceed 1024.

Message ID: 10
Message Type: FATAL/ERROR
Message:
Failed to allocate memory. Check for sufficient memory in your system.

Message ID: 11
Message Type: FATAL
Message:
Failed to create the ROD map file RODMapfileName. Check that you have permissions to create this file.

Extended Help: Stream Out was unable to create a ROD map file. ROD files are created during Stream Out to preserve ROD information. ROD information is generated when you use the Rod Directory option.

Message ID: 12
Message Type: FATAL
Message:

Unable to dump ROD information for the cellview `libraryName/cellName/viewName`.

Extended Help: Stream Out was unable to dump the ROD information for a cellview. ROD files are created during Stream Out to preserve ROD information. ROD information is generated when you use the Rod Director option.

**Message ID: 13**

Message Type: FATAL

Message:

Failed to open the cellview `libraryName/cellName/viewName`. Check that the library exists in the location defined in the `cds.lib` file, and that the cellview exists.

**Message ID: 14**

Message Type: WARNING

Message:

Type `viewType` for cellview `libraryName/cellName/viewName` is not supported by Stream Out. Cellview type `maskLayout` will be used by default.

Extended Help: Stream Out supports only the `dbcMaskLayout` and the `dbcSymbolic` types of cellviews. If a cellview in the design being translated is of any other type, then Stream Out will use `dbcMaskLayout` by default for that cellview.

**Message ID: 15**

Message Type: FATAL

Message:

Invalid orientation for an instance encountered in the cellview `libraryName/cellName/viewName`.

Extended Help: While reading Cadence design database, Stream Out encountered invalid orientation of an instance. The orientations supported by PIPO are `dbcR0`, `dbcR90`, `dbcR180`, `dbcR270`, `dbcMX`, and `dbcMXR90`. 
Message ID: 16

Message Type: WARNING

Message:

Failed to open master cellview `libraryName/cellName/viewName` of a mosaic instance. Check that the library exists in the path defined in the `cds.lib` file and that the cellview exists.

Message ID: 17

Message Type: FATAL

Message:

Invalid orientation encountered for a mosaic instance in the cellview `libraryName/cellName/viewName`. Orientation supported are `dbcR0`, `dbcR90`, `dbcR180`, `dbcR270`, `dbcMX`, and `dbcMXR90`.

Message ID: 18

Message Type: FATAL

Message:

Failed to get column and row spacing for a mosaic instance.

Message ID: 19

Message Type: WARNING

Message:

PIPO does not support attaching text to the object type `objectType`.

Extended Help: PIPO supports text attachment for the rectangle, polygon, path, and line object types only. Text attachment for any other object type will not be translated. However, origin point coordinates of the text would be translated.
Message ID: 22
Message Type: FATAL
Message:
Unknown shape type encountered in the cellview libraryName/cellName/viewName during Stream Out.
Extended Help: Shape types supported by Stream Out are rectangle, polygon, line, path, ellipse, donut, arc, textdisplay, label, and dot.

Message ID: 23
Message Type: FATAL
Message:
Unable to read the bounding box of a rectangle from the cellview libraryName/cellName/viewName.

Message ID: 24
Message Type: FATAL
Message:
Unable to read a polygon from the cellview libraryName/cellName/viewName.

Message ID: 25
Message Type: WARNING
Message:
The SKILL function poUserShapeHandler has modified a polygon at layer layerNumber and startpoint xcoord, ycoord.
Extended Help: If a user defines the SKILL function poUserShapeHandler before running Stream Out, all the shapes present in the input database will pass through this function before being translated to the Stream format. This function might modify or drop the shape that is being translated.
Message ID: 26
Message Type: WARNING
Message:
The SKILL function `poUserShapeHandler` has dropped a polygon at layer `layerNumber` and startpoint `xcoord, ycoord`.

Extended Help: If a user defines the SKILL function `poUserShapeHandler` before running Stream Out, all the shapes present in the input database will pass through this function before being translated to the Stream format. This function might modify or drop the shape that is being translated.

Message ID: 27
Message Type: WARNING
Message:
The SKILL function `poUserShapeHandler` was unable to set the variable `pipoArray` while processing a polygon at layer `layerNumber` and startpoint `xcoord, ycoord`.

Extended Help: If a user defines the SKILL function `poUserShapeHandler` before running Stream Out, all the shapes present in the input database will pass through this function before being translated to the Stream format. This function might modify or drop the shape that is being translated.

Message ID: 28
Message Type: FATAL
Message:
Unable to convert a line to zero-width path in the cellview `libraryName/cellName/viewName`.

Extended Help: When the Convert Lines to option is assigned the value path, line objects are converted to zero-width paths during Stream Out.

Message ID: 30
Message Type: WARNING
Message:

The SKILL function poUserShapeHandler has modified a path at layer layerNumber and startpoint xcoord, ycoord.

Extended Help: If a user defines the SKILL function poUserShapeHandler before running Stream Out, all the shapes present in the input database will pass through this function before being translated to the Stream format. This function might modify or drop the shape that is being translated.

Message ID: 31
Message Type: WARNING

Message:

The SKILL function poUserShapeHandler was unable to set the variable pipoArray while processing a path at layer layerNumber and startpoint xcoord, ycoord.

Extended Help: If a user defines the SKILL function poUserShapeHandler before running Stream Out, all the shapes present in the input database will pass through this function before being translated to the Stream format. This function might modify or drop the shape that is being translated.

Message ID: 32
Message Type: WARNING

Message:

The SKILL function poUserShapeHandler dropped a path at layer layerNumber and startpoint xcoord, ycoord.

Extended Help: If a user defines the SKILL function poUserShapeHandler before running Stream Out, all the shapes present in the input database will pass through this function before being translated to the Stream format. This function might modify or drop the shape that is being translated.

Message ID: 33
Message Type: FATAL

Message:
Unable to read a polygon which was created by converting a path from the cellview
libraryName/cellName/viewName.

Extended Help: The paths are converted to polygons during translation to the Stream format
when the Convert Paths to Polygons option is used.

**Message ID: 34**

Message Type: FATAL

Message: Unable to read a path extension from the cellview libraryName/cellName/viewName.

**Message ID: 35**

Message Type: WARNING

Message: Unknown path style encountered while reading a path from the cellview libraryName/
cellName/viewName.

Extended Help: Stream Out encountered an unknown path style in the input database. Path
styles supported are dbcVarExtendExtend, dbcRoundRound, dbcExtendExtend, and
dbcTruncateExtend.

**Message ID: 36**

Message Type: FATAL

Message: Unable to read a text display from the cellview libraryName/cellName/viewName.

**Message ID: 37**

Message Type: FATAL

Message:
Unknown rotation encountered for a text display in the cellview libraryName/cellName/viewName. Valid rotations are dbcR0, dbcR90, dbcR180, dbcR270, dbcMY, dbcMYR90, dbcMX, and dbcMXR90.

**Message ID: 38**

Message Type: FATAL

Message:

Unknown rotation encountered for a label in the cellview libraryName/cellName/viewName. Valid rotations are dbcR0, dbcR90, dbcR180, dbcR270, dbcMY, dbcMYR90, dbcMX, and dbcMXR90.

**Message ID: 39**

Message Type: WARNING

Message: The user-defined SKILL function poLayerMap() does not map layer-purpose pair layerName:purposeName. All objects on this LPP will be dropped.

**Message ID: 40**

Message Type: FATAL

Message:

Unable to convert a dot from the cellview libraryName/cellName/viewName to a polygon.

Extended Help: Stream Out was unable to convert a dot from the specified cellview to polygon. Dots are converted to polygons during translation when the Convert Dots to option is assigned the value polygon.

**Message ID: 41**

Message Type: FATAL

Message:

Unable to convert a dot from the cellview libraryName/cellName/viewName to a node.
Extended Help: Stream Out was unable to convert a dot from the specified cellview to node. Dots are converted to nodes during translation when the Convert Dots to option is assigned the value node.

Message ID: 42

Message Type: WARNING

Message:

The shape type shapeType in cellview libraryName/cellName/viewName is not supported by Stream Out. It will be dropped.

Extended Help: Shape types supported by Stream Out are rectangle, polygon, line, path, ellipse, donut, arc, text display, label, and dot.

Message ID: 43

Message Type: WARNING

Message:

Unable to flatten the instance instanceName in the cellview libName/cellName/viewName.

Extended Help: Stream Out failed to flatten an instance in the specified cellview. Flattening of pcells takes place when the Convert PCells to Geometry option is used.

Message ID: 44

Message Type: FATAL

Message:

Unable to flatten mosaic mosaicInstanceName in the cellview libName/cellName/viewName.

Extended Help: Stream Out failed to flatten a mosaic in the specified cellview. Flattening of pcells takes place when the Convert PCells to Geometry option is used.
Message ID: 45

Message Type: FATAL

Message:

Stream Out encountered an invalid value for the Convert PCells to Geometry option used in template file. This option has been modified to accept boolean values only. Valid values are true and nil. Update your template file accordingly.

Message ID: 46

Message Type: WARNING

Message:

Options Keep PCells and Convert PCells to Geometry cannot be used together. Turning off the Convert PCells to Geometry option.

Extended Help: The Keep PCells option is used to preserve pcells during translation and the Convert PCells to Geometry option is used to convert pcells to geometries. Pcells cannot be preserved and converted to geometries at the same time. Therefore the Convert Pcells to Geometry option is turned off.

Message ID: 47

Message Type: FATAL

Message:

Unable to read the begin and end path extensions from the input database.

Message ID: 48

Message Type: FATAL

Message:

Unable to read the pin access direction for an instance from the input database.
Message ID: 49
Message Type: WARNING
Message:
Property mapping does not support the object type $objectType$.
Extended Help: Object types supported by property mapping file include rectangle, polygon, dot, line, path, arc label, ellipse, donut, instance, and mosaic. For more information on property mapping, refer to Cadence documentation.

Message ID: 50
Message Type: WARNING
Message:
The property string $stringName$ exceeds the maximum length $maximumlength$ supported by the Stream format. It will be truncated.

Message ID: 51
Message Type: WARNING
Message:
Unable to translate mosaic instance to a Stream array because of orientation error. Check that the orientation of the mosaic instance is correct.

Message ID: 52
Message Type: WARNING
Message:
Unable to create the $pipo_Pcell_info$ file. Check that you have permissions to create this file in the run directory.

Message ID: 53
Message Type: FATAL
Message:
Invalid orientation encountered for mosaic instance in the cellview `libName/cellName/viewName`. Orientations supported by Stream Out are `dbcR0`, `dbcR90`, `dbcR180`, `dbcR270`, `dbcMX`, and `dbcMXR90`.

**Message ID: 54**

Message Type: ERROR

Message:

A mosaic instance `mosaicInstanceName` in the cellview `libName/cellName/viewName` exceeds the row or column GDSII limit of 32767. Break the mosaic instance into two or more instances in the Layout Editor.

**Message ID: 55**

Message Type: FATAL

Message:

Failed to create the `KPDIR/pCellTable` file for dumping pcell data. Check that you have permissions to create this file.

Extended Help: Stream Out was unable to create the file `KPDIR/pCellTable` in the run directory. This file is created during Stream Out to dump the pcell table if the Keep PCells option is used.

**Message ID: 56**

Message Type: FATAL

Message:

Unable to find the `KPDIR/pCellTable` file. Verify that Stream Out was executed with the Keep PCells option in the same directory before executing Stream In.

**Message ID: 58**

Message Type: FATAL
Message:

Failed to open technology file libraryName:techfileName in append mode. Check that the library exists and that you have write permissions to the technology file.

**Message ID: 59**

Message Type: FATAL

Message:

A technology library is required for preserving pcells (using the Keep PCells option). Specify a technology library with its correct location by using the Stream In Options form (or by setting the attachTechfileOfLib option in the template file). You can also attach a technology library to the target design library. In case you receive this message even when a technology file exists at the specified location or a technology file is already attached to the target library, the technology library might be corrupt. In that case, try recreating the technology library using Stream Out.

**Message ID: 60**

Message Type: FATAL

Message:

Technology file kp_techfile_devices.tf located in KPDIR directory is not readable. Check that this file exists and you have read permissions to this file.

**Message ID: 61**

Message Type: WARNING

Message:

Failed to open the pcell supermaster cellview cellName/viewName from reference library libraryName. Creating cellview in the target library libraryName.

Extended Help: Stream In failed to open a pcell super master cellview from a reference library. Check that reference library exists in the location defined in the cds.lib file and that the cellview also exists in the reference library.
Message ID: 62
Message Type: FATAL
Message:
Library name libraryName exceeds maximum length libraryLength supported during Stream In.

Message ID: 63
Message Type: FATAL
Message:
Library name libraryName specified in the Attach Techfile of Library option exceeds maximum length libraryLength supported during Stream In.

Message ID: 64
Message Type: FATAL
Message:
Library name libraryName specified in the Attach Techfile of Library option does not exist. Check that the library name and its path are correctly defined in the cds.lib file.

Message ID: 65
Message Type: FATAL
Message:
Failed to open technology file techfileName in read mode from the library libraryName. This library is specified in the Attach Techfile of Library option. Check that you have permissions to read the technology file.

Message ID: 66
Message Type: FATAL
Message:
Failed to create target library `libraryName` in the run directory. Check that you have permissions to create the target library.

**Message ID: 67**

Message Type: WARNING

Message:

Undefined layer-purpose pairs will not be translated because Stream In does not update the attached technology file.

Extended Help: You have specified a technology file to be attached to the target library by using the Attach Techfile of Library option. This file is opened in read-only mode during Stream In because other users might be using the same file simultaneously. Any change in the technology file during Stream In might hamper the work of other users. Therefore this file will not be updated with the undefined layer-purpose pairs found in Stream file. Undefined layer-purpose pairs will not be translated.

**Message ID: 68**

Message Type: WARNING

Message:

Failed to attach the technology file from the `libraryName` library because a technology file already exists in the target library.

Extended Help: You have specified a technology file to be attached to the target library by using the Attach Techfile of Library option. This file will not be attached to the target library because a technology file already exists in the target library.

**Message ID: 69**

Message Type: WARNING

Message:

Failed to open the technology file `techfileName` from the library `libraryName` in the append mode. Undefined layer-purpose pairs will not be translated.
Message ID: 70
Message Type: WARNING
Message:
Failed to set value value1 for DBUPerUU variable in the target library libraryName.

Message ID: 71
Message Type: WARNING
Message:
Failed to set value userUnitValue for user unit in the target library libraryName.

Message ID: 72
Message Type: WARNING
Message:
Ignored the ASCII Technology File Name option because you have also used the Attach Techfile of Library option.

Extended Help: During Stream In, if you specify values for both the Attach Techfile of Library and the ASCII Technology File Name options, then the value specified in the ASCII Technology File Name option will be ignored. This is because only one technology file is required. The technology file present in the library that is specified in the Attach Techfile of Library option will be attached to the target library.

Message ID: 73
Message Type: FATAL
Message:
Failed to load ASCII technology file techfileName specified in the ASCII Technology File Name option.
Message ID: 74
Message Type: WARNING

Message:
Failed to read the ASCII technology file techfileName specified in the ASCII Technology File Name option. Check that this file exists in the run directory and that you have permissions to read it.

Message ID: 75
Message Type: WARNING

Message:
Failed to create a cell in the target library because the cell name cellName exceeds maximum length cellNameLength supported during Stream In. Map the long cell name to the short cell name in the Cell name map file.

Message ID: 76
Message Type: FATAL

Message:
Failed to create the cellview libName/cellName/viewName. Check that you have permissions to create this cellview.

Extended Help: Stream In was unable to create a cellview in the target library. It might be because disk space is full or the number of subdirectories have exceeded the maximum limit.

Message ID: 78
Message Type: WARNING

Message:
Failed to set the value of variable minGridResolution for the view viewName.
Message ID: 79
Message Type: FATAL
Message:
Failed to open the ROD map file RODMapfileName during Stream In. ROD file might be missing. Run Stream Out using the Rod Directory option to create the ROD file.

Message ID: 80
Message Type: FATAL
Message:
Failed to create a rectangle at layer layerNumber and bounding box (xcoord, ycoord), (xcoord, ycoord).

Message ID: 81
Message Type: FATAL
Message:
Failed to create a dot at layer layerNumber and coordinates (xcoord, ycoord).

Message ID: 82
Message Type: WARNING
Message:
Failed to retrieve valid pin information from the value of attribute attributeNumber present in the Stream file.

Extended Help: Value of attribute specified in the Restore Pin Attribute option does not have a valid pin information. Check that you have used the same attribute number to preserve pin information at the time of Stream Out.

Message ID: 83
Message Type: WARNING
Message:

Failed to create property propertyName=propertyValue in the target library. This property is retrieved from the value of attribute attributeValue present in the Stream file.

**Message ID: 84**

Message Type: WARNING

Message:

Failed to create a polygon during Stream In because the number of points in the polygon is less than four.

**Message ID: 85**

Message Type: FATAL

Message:

Failed to create a polygon at layer layerNumber and coordinates (xcoord, ycoord).

**Message ID: 86**

Message Type: WARNING

Message:

Failed to create an ellipse at layer layerNumber and bounding box (xcoord, ycoord), (xcoord, ycoord).

**Message ID: 87**

Message Type: FATAL

Message:

Failed to create line for a zero-width path during Stream In. The zero-width path is converted to a line when the Convert Zero Width Paths to option is assigned the value line.
Message ID: 88
Message Type: WARNING
Message:
Ignored a zero-width path during Stream In because the Convert Zero Width Paths to option is assigned the value ignore.

Message ID: 89
Message Type: WARNING
Message:
Stream path record with illegal path type pathType encountered during Stream In. This path record will be ignored.
Extended Help: Stream In ignores the path with invalid Stream type. The valid path types supported by Stream In are 0-5.

Message ID: 90
Message Type: FATAL
Message:
Failed to create path at layer layerNumber and coordinates (xcoord, ycoord) in the target library.

Message ID: 91
Message Type: WARNING
Message:
Failed to set variable extensions for the path at layer layerNumber and coordinates (xcoord, ycoord) in the target library.

Message ID: 92
Message Type: WARNING
Message:

Illegal angle \textit{angle1} encountered in a TEXT record during Stream In. TEXT record will be ignored.

Extended Help: Valid angles for text, supported by PIPO are 0, 90, 180, 270.

**Message ID: 93**

Message Type: WARNING

Message:

A Stream TEXT record having multiline text is encountered. Multiline text will be translated as a single line.

**Message ID: 94**

Message Type: FATAL

Message:

Failed to create label \textit{labelName} at layerNumber and coordinates \textit{xcoord}, \textit{ycoord}.

**Message ID: 95**

Message Type: ERROR

Message:

Failed to create an instance because master cell name \textit{cellName} exceeds maximum length \textit{cellLength} supported during Stream In. Use the Cell name map file to map such cell names.

**Message ID: 96**

Message Type: WARNING

Message:

SREF record in the Stream file contains illegal angle \textit{angle1}. Instance will be dropped.
Extended Help: The valid angles required to create an instance during Stream In are 0, 90, 180, and 270.

**Message ID: 98**

Message Type: WARNING

Message:

A dummy instance of master cell *instanceName* will be created if master cell is not found in Stream file.

Extended Help: Stream In is unable to find the master cell of an instance in the reference and target libraries. During Stream In, data in the Stream file is read sequentially. Till this stage, master cell has not been found. It might be present in the remaining portion of the Stream file that has not been currently read. A dummy instance will be created if the master cell is not found in the Stream file.

**Message ID: 99**

Message Type: WARNING

Message:

Failed to create an instance because the master cell *cellName* was not found in the Stream file.

Extended Help: A master cell is not found in the Stream file because it does not contain complete design. It might be possible that during Stream Out, you have used the **Retain Reference Library (No Merge)** option. Therefore, data in the reference libraries was not translated to the Stream file.

**Message ID: 100**

Message Type: WARNING

Message:

Failed to create via instance *instanceName* of master cell *cellName*.
Message ID: 101
Message Type: FATAL
Message:
Failed to create the instance instanceName of master cell cellName.

Message ID: 102
Message Type: FATAL/WARNING
Message:
Failed to create an array instance because AREF record in the stream file has illegal angle angle. Valid angles supported during Stream In are 0, 90, 180, 270.

Message ID: 103
Message Type: WARNING
Message:
An instance for master cell instanceName is created by ignoring the instance name.

Extended help: Stream In failed to create an instance with the specified name because the instance with that name probably existed in the cellview. Due to this, Stream In will try to create instance without specifying the instance name. In such a case, a unique name will be assigned to the instance by the database.

Message ID: 104
Message Type: WARNING
Message:
Failed to create a simple mosaic instance for the master cell instanceName.

Message ID: 105
Message Type: INFORMATIONAL
Message:

A mosaic instance for the master cell cellName is created by ignoring the mosaic name.

Extended Help: Stream In failed to create a mosaic instance with the specified name because the mosaic instance with the same name probably existed in the cellview. Therefore, Stream In will try to create mosaic instance without specifying the mosaic name. In such a case, a unique name will be assigned to the instance by the database.

**Message ID: 106**

Message Type: FATAL

Message:

Failed to save the cellview cellName/viewName in the target library.

**Message ID: 107**

Message Type: FATAL

Message:

Failed to close the cellview cellName/viewName in the target library.

**Message ID: 108**

Message Type: FATAL

Message:

Failed to save technology file for the target library.

**Message ID: 109**

Message Type: WARNING

Message:

Failed to get the default technology file. Check that default technology file exists in:

installationDirectory/tools/dfII/etc/cdsDefTechLib.
**Message ID: 110**

Message Type: FATAL

Message:

Number of purposes in technology file for layer *layerName* exceed the limit *maxPurpose*.

**Message ID: 111**

Message Type: WARNING

Message:

Invalid Stream layer *layerNumber* defined in the *streamLayers* rule for layer-purpose pair *layerName:* *purposeName*. The valid Stream layer number range is 0-255.

Extended Help: During Stream Out, *streamLayers* rules containing invalid Stream layer or Stream datatype in the technology file will be ignored. Due to this, all objects on the Cadence layer that is mapped to the invalid Stream layer or Stream datatype will also be ignored.

During Stream In, when a Stream layer or a Stream datatype greater than 255 is encountered in the *streamLayers* rules in the technology file, a warning is displayed but all the objects on this Stream layer or Stream datatype are translated. Stream layers or Stream datatypes greater than 255 are translated to support Stream files created by other vendors. However, the negative Stream layers or Stream datatypes are not supported and their objects will not be translated.

**Message ID: 112**

Message Type: WARNING

Message:

Invalid Stream datatype *streamdataType* defined in the *streamLayers* rule for layer-purpose pair *layerName:* *purposeName*. The valid Stream datatype range is 0-255.

Extended Help: During Stream Out, *streamLayers* rules containing invalid Stream layer or Stream datatype in the technology file will be ignored. Due to this, all objects on the Cadence layer that is mapped to the invalid Stream layer or Stream datatype will also be ignored.

During Stream In, when a Stream layer or a Stream datatype greater than 255 is encountered in the *streamLayers* rules in the technology file, a warning is displayed but all the objects on this Stream layer or Stream datatype are translated. Stream layers or Stream datatypes...
greater than 255 are translated to support Stream files created by other vendors. However, the negative Stream layers or Stream datatypes are not supported and their objects will not be translated.

**Message ID: 113**

Message Type: WARNING

Message:

Non-integer Stream layer defined in the streamLayers rule for layer-purpose pair layerName:purposeName will be skipped.

Extended Help: The streamLayers rule for a layer defined in the technology file contains a non-integer Stream layer number. Therefore, all the shapes on this layer will be ignored during Stream In. The valid Stream layer numbers range is 0-255.

**Message ID: 114**

Message Type: WARNING

Message:

Non-integer Stream datatype defined in streamLayers rule for layer-purpose pair layerName:purposeName will be skipped.

Extended Help: The streamLayers rule for a layer defined in the technology file contains a non-integer Stream datatype. Therefore, all the shapes on this layer will not be translated during Stream In. The valid Stream layer numbers range is 0-255.

**Message ID: 115**

Message Type: FATAL

Message:

The number of layer-purpose pairs skipped exceeds maximum number that Stream In can handle.

Extended Help: During Stream In, maximum of 64 layers are supported, and for each individual layer 64 purposes can be handled. Therefore total number of layer-purpose pairs supported by Stream In are 4096 (64 X 64). During Stream In more than 4096 undefined
layer-purpose pairs were skipped. It means that number of layer-purpose pairs used in input Stream file exceed the maximum number that can be handled.

**Message ID: 116**

Message Type: WARNING

Message:

Skipped undefined layer-purpose pair `layerName:purposeName`.

Extended Help: Stream In skipped a layer encountered in Stream file which is not defined in technology file. All the shapes on this layer-purpose pair will not be translated. This occurs when the Skip Undefined Layer-Purpose Pair option is used or if the technology file from other technology library is attached to target library.

**Message ID: 117**

Message Type: WARNING

Message:

Reserved system layer `layerNumber` number encountered in the Stream file. It will be skipped.

Extended Help: Layer numbers greater than 194 are considered as reserved layers in Cadence database. Reserved layers encountered in Stream file will not be translated during Stream In. Layer map file can be used to map reserved layers to non reserved layers. For more information, refer to Cadence documentation.

**Message ID: 118**

Message Type: WARNING

Message:

Failed to create the new layer-purpose pair `layerName:purposeName`. It will be skipped.

**Message ID: 119**

Message Type: WARNING
Message:

Failed to set the visibility attribute of a newly created layer-purpose pair 
layerName:purposeName.

Extended Help: Stream In failed to set the visibility attribute of a newly created layer-purpose pair. A new layer-purpose pair is created in the technology file when a layer-purpose pair used in Stream file is not defined in technology file, and you are not using the Skip Undefined Layer-Purpose Pair option.

Message ID: 120

Message Type: WARNING

Message:

Failed to set the priority of a newly created layer-purpose pair 
layerName: purposeName.

Extended Help: Stream In failed to set the priority of a newly created layer-purpose pair. A new layer-purpose pair is created in the technology file when a layer-purpose pair used in Stream file is not defined in technology file, and you are not using the Skip Undefined Layer-Purpose Pair option.

Message ID: 121

Message Type: WARNING

Message:

Failed to update the packet name of a newly created layer-purpose pair 
layerName: purposeName.

Extended Help: Stream In failed to update the packet name of a newly created layer-purpose pair. A new layer-purpose pair is created in the technology file when a layer-purpose pair used in Stream file is not defined in technology file, and you are not using the Skip Undefined Layer-Purpose Pair option.

Message ID: 122

Message Type: INFORMATIONAL

Message:
Creating a new layer-purpose pair \textit{layerName:purposeName} in the technology file.

Extended Help: A new layer-purpose pair is created in the technology file when a layer-purpose pair used in the Stream file is not defined in the technology file, and you are not using the Skip Undefined Layer-Purpose Pair option.

**Message ID: 123**

Message Type: WARNING

Message:

Cell name \textit{cellName} has illegal characters. It will be changed to \textit{cellName}.

Extended Help: Stream In replaces an illegal character by \_ in a cell name. The legal cell names supported by Stream In are: (a-z, A-Z, 0-9,., _, -, +, /, $,?,%, &, #).

**Message ID: 124**

Message Type: WARNING

Message:

Failed to delete the cellview \textit{cellName/viewName} from the target library.

**Message ID: 125**

Message Type: FATAL

Message:

Failed to find library \textit{libraryName} specified in the Reference Library Order option.

**Message ID: 126**

Message Type: WARNING

Message:

The \texttt{skipPcdbGen} option is chosen. The generation of pc.db files in the output cellviews in the Cadence database will be skipped.
Extended Help: To enhance the performance of Stream In process you can use skipPcdbGen option. The pc.db files are not created in the output cellviews in Cadence database during Stream In when you use the skipPcdbGen template file option. The absence of pc.db files in the CDBA database might result in some problems while using the Hierarchy Browser tool or the Layout Versus Schematic (LVS) tool at times.

**Message ID: 127**

**Message Type: WARNING**

**Message:**

The skipDbLocking option is chosen. The output cellviews in the Cadence database will no longer be separately locked while being edited during Stream In.

Extended Help: To enhance the performance of Stream In process you can use skipDbLocking option. During Stream In, database locking is switched off when the skipDbLocking template file option is used. While using this option, ensure that the target library is not accessed by another user simultaneously. This needs to be done to prevent data corruption if a cellview is accessed by Stream In as well as by another user, or any other process at the same time.

**Message ID: 128**

**Message Type: WARNING**

**Message:**

The skipPcdbGen option has been turned off because the target library libraryName already exists.

Extended Help: To enhance the performance of Stream In process you can use the skipPcdbGen option. During Stream In, the pc.db files are not created in the output cellviews in Cadence database when the skipPcdbGen template file option is used. If the target library exists, then this option is automatically turned off to avoid inconsistency in the target library.

**Message ID: 129**

**Message Type: WARNING**

**Message:**
The `skipDbLocking` option has been turned off because the target library `libraryName` already exists.

Extended Help: To enhance the performance of Stream In process you can use `skipDbLocking` option. During Stream In, database locking is switched off when the `skipDbLocking` template file option is used. If the target library exists then this option is automatically turned off to avoid data corruption because the target library might be in use by some other process.

**Message ID: 130**

Message Type: FATAL

Message:

Failed to write a Stream record to disk. Check that there is sufficient disk space and the file size limit is not exceeded.

**Message ID: 131**

Message Type: FATAL

Message:

Failed to open the input Stream file `fileName`. Check that the Stream file exists and is a regular UNIX file.

**Message ID: 132**

Message Type: WARNING

Message:

Failed to get integer value from a Stream record because the data length is negative.

**Message ID: 133**

Message Type: WARNING

Message:
Failed to get an integer from a Stream record because invalid integer datatype `dataType` was encountered in the Stream record.

Extended Help: Stream In failed to get an integer from a Stream record because it does not have a valid integer datatype. Valid datatypes for integer values are: 1 (for integer array), 2 (for two byte integer), 3 (for four-byte integer).

**Message ID: 134**

**Message Type: FATAL**

**Message:**

Failed to create Stream file `fileName` in the run directory. Check that you have permission to create this file and that there is sufficient disk space.

**Message ID: 135**

**Message Type: WARNING**

**Message:**

Invalid Stream version number `versionNumber` is specified in the template file. Using default Stream version number 5.

**Message ID: 136**

**Message Type: FATAL**

**Message:**

Bad library name `libraryName` specified in GUI or template file.

**Message ID: 137**

**Message Type: WARNING**

**Message:**

Layer number `layerNumber` is out of range 0-maximumRange.
Extended Help: The GDSII format supports layer numbers in the range 0-63. During Stream Out, if you use the Respect GDSII limits option, then a warning will be issued for any layer number greater than 63. If the Respect GDSII limits option is not used, a warning will be issued for any layer number greater than 255.

Message ID: 138

Message Type: WARNING

Message:

Total length of rectangle properties exceed $rectangleLength$ and the extra properties will not be preserved.

Extended Help: During Stream Out, if the Respect GDSII limits option is used and the total length of properties to be preserved for an object exceeds 128, the extra properties will be dropped. If the Respect GDSII limits option is not used, the properties will be dropped when the total length exceeds 1024.

Message ID: 139

Message Type: WARNING

Message:

A polygon having number of vertexes $totalVertexes$ is written to the Stream file. Maximum number of vertexes supported by GDSII format are $maxVertexes$.

Extended Help: During Stream Out, a polygon with number of vertexes exceeding the maximum number allowed by Stream format was written to the Stream file. You can control the maximum vertexes in a polygon by using the Maximum Vertices in Path/Polygon option. Stream Out breaks a large polygon into smaller polygons so that total number of vertexes written to the Stream file does not exceed the value specified in the Maximum Vertices in Path/Polygon option.

Message ID: 140

Message Type: WARNING

Message:

The total length of polygon properties exceed $maxPropertyLength$ and the extra properties will not be preserved.
Extended Help: During Stream Out, if the Respect GDSII limits option is used and the total length of properties to be preserved for an object exceeds 128, the extra properties will be dropped. If the Respect GDSII limits option is not used, the properties will be dropped when total length exceeds 1024.

**Message ID: 141**

Message Type: WARNING

Message:

The total length of node properties exceed \textit{maxPropertyLength} and the extra properties will not be preserved.

Extended Help: During Stream Out, if the Respect GDSII limits option is used and the total length of properties to be preserved for an object exceeds 128, the extra properties will be dropped. If the Respect GDSII limits option is not used, the properties will be dropped after total length exceeds 1024.

**Message ID: 142**

Message Type: WARNING

Message:

The total length of path properties exceed \textit{maxPropertyLength} and the extra properties will not be preserved.

Extended Help: During Stream Out, if the Respect GDSII limits option is used and the total length of properties to be preserved for an object exceeds 128, the extra properties will be dropped. If the Respect GDSII limits option is not used, the properties will be dropped after total length exceeds 1024.

**Message ID: 143**

Message Type: WARNING

Message:

The total length of text properties exceed \textit{maxPropertyLength} and the extra properties will not be preserved.
Extended Help: During Stream Out, if the Respect GDSII limits option is used and the total length of properties to be preserved for an object exceeds 128, the extra properties will be dropped. If the Respect GDSII limits option is not used, the properties will be dropped after total length exceeds 1024.

**Message ID: 144**

Message Type: WARNING

Message:

Total length of instance properties exceed $maxPropertyLength$ and the extra properties will not be preserved.

Extended Help: During Stream Out, if the Respect GDSII limits option is used and the total length of properties to be preserved for an object exceeds 128, the extra properties will be dropped. If the Respect GDSII limits option is not used, the properties will be dropped after total length exceeds 1024.

**Message ID: 145**

Message Type: WARNING

Message:

The number of vertexes $totalVertexes$ in a node exceed the GDSII limit $maxVertexes$. Node will be ignored.

**Message ID: 146**

Message Type: WARNING

Message:

Writing a path with number of vertexes $totalVertexes$ that exceed the GDSII limit $maxVertexes$.

Extended Help: During Stream Out, a path with number of vertexes exceeding the maximum number allowed by Stream format was written to the Stream file. You can control the maximum vertexes in a path by using the Maximum Vertices in Path/Polygon option. Stream Out breaks a large path into smaller paths so that total number of vertexes written in the Stream file do not exceed the value specified in the Maximum Vertices in Path/Polygon option.
Message ID: 147
Message Type: WARNING
Message:
The Cadence font `CadenceFontType` for string `text` is mapped to Stream font `StreamFontType`.

Extended Help: During Stream Out the Cadence fonts `Stick`, `EuroStyle`, `GothicFont`, and `MathFont` are mapped to the Stream fonts 0,1,2,3 respectively. Any other Cadence font is mapped to Stream font 0.

Message ID: 148
Message Type: WARNING
Message:
A duplicate cellname `cellName` found in the Stream file. It will be ignored.

Message ID: 149
Message Type: FATAL
Message:
Bad Stream record type `recordType` encountered in the input Stream file. Valid range is 0-59.

Message ID: 150
Message Type: WARNING
Message:
Failed to get string corresponding to name of a structure from the Stream record type `STRM_STRNAME` (6).

Extended Help: PIPO failed to get a string corresponding to name of a structure from the Stream record type `STRM_STRNAME` (6) because the record is corrupt. The `SREF` construct containing this record will not be translated.
**Message ID: 151**

Message Type: FATAL

Message:

Failed to get integer corresponding to the Stream version from the Stream record type `STRM_HEADER (0)` because the record is corrupt.

**Message ID: 152**

Message Type: FATAL

Message:

Failed to open input Stream file `fileName`. Check that the Stream file exists and that you have read permission to this file.

**Message ID: 153**

Message Type: FATAL

Message:

Invalid record sequence encountered in the input Stream file. First record should be of type `STRM_HEADER (0)`.

**Message ID: 155**

Message Type: FATAL

Message:

Bad Stream record type sequence `recordType1, recordType2` encountered in the input Stream file.

Extended Help: The input Stream file does not have correct record sequence because Stream record type `STRM_BGNLIB` is not present after Stream record type `STRM_HEADER (0)`. The Stream record type `STRM_BGNLIB (1)` should be present immediately after the Stream record type `STRM_HEADER (0)`.
Message ID: 156
Message Type: FATAL
Message:
Failed to get integer corresponding to time from the Stream record type STRM_BGNLIB (1). Data in the input Stream file might be corrupt.

Message ID: 157
Message Type: WARNING
Message:
Years in non-standard two-digit format encountered in the Stream file.

Extended Help: The Stream record of type STRM_BGNLIB (1) has information pertaining to years in non-standard format. The input year is expected to be the number of years since 1900. If the input year is between 00 and 69, it is assumed that this year is in two-digit format and represents a year between 2000 and 2069. For more information, refer to Cadence documentation.

Message ID: 158
Message Type: WARNING
Message:
Years in non-standard four-digit format encountered in the Stream file.

Extended Help: The Stream record of type STRM_BGNLIB (1) has information pertaining to years in non-standard format. The input year is expected to be the number of years since 1900. If the input year is greater than 1970, it is assumed that the year is in four-digit format and represent the actual year concerned. 1900 is subtracted from it to get the number of years since 1900. For more information, refer to Cadence documentation.

Message ID: 160
Message Type: FATAL
Message:
The record length \textit{recordLength} for the Stream record type \texttt{STRM_REFLIBS (31)} exceeds the limit \textit{maximumLimit}. Reference library name will be truncated.

\textbf{Message ID: 161}

Message Type: WARNING

Message:

Failed to get string corresponding to the reference library names from the Stream record type \texttt{STRM_REFLIB (31)} because the record is corrupt.

\textbf{Message ID: 162}

Message Type: INFORMATIONAL

Message:

The reference library \textit{libraryName} defined in the Stream file will be ignored because the Retain Reference Library (No Merge) option is not used.

\textbf{Message ID: 163}

Message Type: FATAL

Message:

Failed to get a real number corresponding to the units from the Stream record type \texttt{STRM_UNITS (3)} because the record is corrupt.

\textbf{Message ID: 164}

Message Type: FATAL/WARNING

Message:

Bad unit information found in the Stream file. \texttt{U-UNIT/DBU: userUnitperDBUvalue, METRIC/DBU: metricperDBUvalue}.

Extended Help: The record of type \texttt{STRM_UNITS (3)} present in the input Stream file contains bad unit information. The desired value of \texttt{U-UNIT/DBU} and \texttt{METRIC/DBU} are such that the
ratio (U-UNIT/DBU) / (METRIC/DBU) should have one of the following values: 1000000 (microns), 39370.1 (mils), 39.3701 (inches), 1000 (mm), 100 (cm), 1024000 (microns).

**Message ID: 165**

Message Type: WARNING

Message:

Unexpected Stream record type `recordType` encountered in the library record.

Extended Help: The input Stream file contains an invalid record sequence. Only the following record types can occur between the Stream record type `STRM_BGNLIB (1)` and `STRM_ENDLIB (4)`: `STRM_LIBDIRSIZE (57)`, `STRM_SRFNAME (58)`, `STRM_LIBSECUR (59)`, `STRM_LIBNAME (2)`, `STRM_REFLIBS (31)`, `STRM_FONTS (32)`, `STRM_ATTRTABLE (35)`, `STRM_GENERATIONS (34)`, `STRM_FORMAT (54)`, `STRM_MASK (55)`, `STRM_ENDMASKS (56)`, `STRM_UNITS (3)`, `STRM_BGNSTR (5)`.

**Message ID: 166**

Message Type: FATAL

Message:

End of file was encountered before the Stream record type `STRM_ENDLIB (4)` in the Stream file.

Extended Help: The input Stream file contains invalid record sequence. It looks as if none of the Stream data has been read at all. This problem might be occurring because data is not a valid Stream format file. The Stream file might also be in the swapped byte format. If yes, use the Unix `dd` utility with the `conv=swab` option to correct the byte ordering.

**Message ID: 167**

Message Type: WARNING

Message:

Failed to get a string corresponding to name of a structure from the Stream record type `STRM_XY (16)`, because the record is corrupt. This structure will not be translated.
Message ID: 168
Message Type: WARNING
Message:
Unexpected Stream record type \textit{recordType} encountered in the structure.

Extended Help: The input Stream file contains invalid record sequence. Only the following record types can occur between the Stream record type \textit{STRM\_BGNSTR (5)} and \textit{STRM\_ENDEL (7)} - \textit{STRM\_STRNAME (6), STRM\_BOUNDARY (8), STRM\_PATH (9), STRM\_SREF (10), STRM\_AREF (11), STRM\_TEXT (12), STRM\_NODE (21), STRM\_BOX (45)}.

Message ID: 169
Message Type: WARNING
Message:
Unexpected Stream record type \textit{recordType} encountered in the boundary record.

Extended Help: The input Stream file contains an invalid record sequence. A Stream record type less than 12 cannot exist between the Stream record type \textit{STRM\_BOUNDARY (8)} and \textit{STRM\_ENDEL (17)}.

Message ID: 170
Message Type: WARNING
Message:
Invalid layer $layerNumber$ encountered in the Stream file. Valid range is 0-$maximumLimit$.

Message ID: 171
Message Type: WARNING
Message:
Failed to get an integer corresponding to a boundary layer from the Stream record type \textit{STRM\_LAYERS (13)}.
Extended Help: PIPO failed to get an integer corresponding to a boundary layer from the record type **STRM_LAYER (13)** because the record is corrupt. The **BOUNDARY** construct containing this record will not be translated.

**Message ID: 172**
Message Type: WARNING
Message:
Failed to get an integer corresponding to a path layer from the Stream record type **STRM_LAYER (13)**.

Extended Help: PIPO failed to get an integer corresponding to a boundary layer from the record type **STRM_LAYER (13)** because the record is corrupt. The **PATH** construct containing this record will not be translated.

**Message ID: 173**
Message Type: WARNING
Message:
Failed to get an integer corresponding to a box layer from the Stream record type **STRM_LAYER (13)**.

Extended Help: PIPO failed to get an integer corresponding to a box layer from the record type **STRM_LAYER (13)** because the record is corrupt. The **BOX** construct containing this record will not be translated.

**Message ID: 174**
Message Type: WARNING
Message:
Failed to get an integer corresponding to a node layer from the Stream record type **STRM_LAYER (13)**.

Extended Help: PIPO failed to get an integer corresponding to a node layer from the record type **STRM_LAYER (13)** because the record is corrupt. The **NODE** construct containing this record will not be translated.
**Message ID: 175**

Message Type: WARNING

Message:

Failed to get an integer corresponding to a text layer from the Stream record type STRM_LAYER (13).

Extended Help: PIPO failed to get an integer corresponding to a text layer from the record type STRM_LAYER (13) because the record is corrupt. The TEXT construct containing this record will not be translated.

**Message ID: 176**

Message Type: WARNING

Message:

Failed to get an integer corresponding to a boundary datatype from the Stream record type STRM_DATATYPE (14).

Extended Help: PIPO failed to get an integer corresponding to a boundary datatype from the Stream record type STRM_DATATYPE (14) because the record is corrupt. The BOUNDARY construct containing this record will not be translated.

**Message ID: 177**

Message Type: WARNING

Message:

Failed to get an integer corresponding to a path datatype from the Stream record type STRM_DATATYPE (14).

Extended Help: PIPO failed to get an integer corresponding to a path datatype from the Stream record type STRM_DATATYPE (14) because the record is corrupt. The PATH construct containing this record will not be translated.

**Message ID: 178**

Message Type: WARNING
Message:

Failed to get an integer corresponding to a box type from the Stream record type STRM_BOXTYPE (46).

Extended Help: PIPO failed to get an integer corresponding to a box datatype from the Stream record type STRM_BOXTYPE (46) because the record is corrupt. The BOX construct containing this record will not be translated.

**Message ID: 179**

**Message Type: WARNING**

Message:

Failed to get an integer corresponding to a nodetype from the Stream record type STRM_NODETYPE (42).

Extended Help: PIPO failed to get an integer corresponding to a nodetype from the Stream record type STRM_NODETYPE (42) because the record is corrupt. The NODE construct containing this record will not be translated.

**Message ID: 180**

**Message Type: WARNING**

Message:

Failed to get an integer corresponding to a text datatype from the Stream record type STRM_TEXTTYPE (22).

Extended Help: PIPO failed to get an integer corresponding to a text datatype from the Stream record type STRM_TEXTTYPE (22) because the record is corrupt. The TEXT construct containing this record will not be translated.

**Message ID: 181**

**Message Type: WARNING**

Message:

The number of vertexes $num\text{Vertexes}$ in a boundary is out of range $4\text{ }max\text{Vertexes}$ supported by the Cadence database.
Message ID: 182

Message Type: WARNING

Message:

Failed to get an integer array corresponding to the vertexes of a boundary from the Stream record type STRM_XY (16).

Extended Help: PIPO failed to get an integer array corresponding to the vertexes of a boundary from the Stream record type STRM_XY (16) because the record is corrupt. The BOUNDARY construct containing this record will not be translated.

Message ID: 183

Message Type: WARNING

Message:

Failed to get an integer array corresponding to the vertexes of a PATH from the Stream record type STRM_XY (16).

Extended Help: PIPO failed to get an integer array corresponding to the vertexes of a path from the Stream record type STRM_XY (16) because the record is corrupt. The PATH construct containing this record will not be translated.

Message ID: 184

Message Type: WARNING

Message:

Failed to get an integer array corresponding to the vertexes of a node from the Stream record type STRM_XY (16).

Extended Help: PIPO failed to get an integer array corresponding to the vertexes of a node from the Stream record type STRM_XY (16) because the record is corrupt. The NODE construct containing this record will not be translated.

Message ID: 185

Message Type: WARNING
Message:

Failed to get an integer array corresponding to the vertexes of a box from the Stream record type STRM_XY (16).

Extended Help: PIPO failed to get an integer array corresponding to the vertexes of a box from the Stream record type STRM_XY (16) because the record is corrupt. The BOX construct containing this record will not be translated.

Message ID: 186

Message Type: WARNING

Message:

In a boundary record the first xcoord, ycoord and the last xcoord, ycoord vertexes are not same.

Extended Help: The input Stream file contains a corrupt Stream record of type STRM_XY (16) in the boundary record. The first and last vertexes of a boundary are not same. This boundary record will not be translated.

Message ID: 187

Message Type: WARNING

Message:

Failed to get an integer corresponding to a boundary attribute number from the Stream record type STRM_PROPATTR (43).

Extended Help: PIPO failed to get an integer corresponding to the boundary attribute number from the Stream record type STRM_PROPATTR (43) because the record is corrupt. The BOUNDARY construct containing this record will not be translated.

Message ID: 188

Message Type: WARNING

Message:

Failed to get an integer corresponding to the path attribute number from the Stream record type STRM_PROPATTR (43).
Extended Help: PIPO failed to get an integer corresponding to the path attribute number from the Stream record type STRM_PROPATTR (43) because the record is corrupt. The PATH construct containing this record will not be translated.

Message ID: 189
Message Type: WARNING
Message:

Failed to get an integer corresponding to the box attribute number from the Stream record type STRM_PROPATTR (43).

Extended Help: PIPO failed to get an integer corresponding to the box attribute number from the Stream record type STRM_PROPATTR (43) because the record is corrupt. The BOX construct containing this record will not be translated.

Message ID: 190
Message Type: WARNING
Message:

Failed to get an integer corresponding to the node attribute number from the Stream record type STRM_PROPATTR (43).

Extended Help: PIPO failed to get an integer corresponding to the node attribute number from the Stream record type STRM_PROPATTR (43) because the record is corrupt. The NODE construct containing this record will not be translated.

Message ID: 191
Message Type: WARNING
Message:

Failed to get an integer corresponding to the cell attribute number from the Stream record type STRM_PROPATTR (43).

Extended Help: PIPO failed to get an integer corresponding to the cell attribute number from the Stream record type STRM_PROPATTR (43) because the record is corrupt. The SREF construct containing this record will not be translated.
Message ID: 192

Message Type: WARNING

Message:

Failed to get an integer corresponding to the text attribute number from the Stream record type STRM_PROPATTR (43).

Extended Help: PIPO failed to get an integer corresponding to the text attribute number from the Stream record type STRM_PROPATTR (43) because the record is corrupt. The TEXT construct containing this record will not be translated.

Message ID: 193

Message Type: WARNING

Message:

Failed to get an integer corresponding to the array attribute number from the Stream record type STRM_PROPATTR (43).

Extended Help: PIPO failed to get an integer corresponding to the array attribute number from the Stream record type STRM_PROPATTR (43) because the record is corrupt. The AREF construct containing this record will not be translated.

Message ID: 194

Message Type: WARNING

Message:

Length `propertyLength` of a property attached to a boundary record exceeds the limit `maximumLimit`. This property will be dropped.

Message ID: 195

Message Type: WARNING

Message:

Length `propertyLength` of a property attached to a path record exceeds the limit `maximumLimit`. This property will be dropped.
**Message ID: 196**

Message Type: WARNING

Message:

Length `propertyLength` of a property attached to a box record exceeds the limit `maximumLimit`. This property will be dropped.

**Message ID: 197**

Message Type: WARNING

Message:

Length `propertyLength` of a property attached to a node record exceeds the limit `maximumLimit`. This property will be dropped.

**Message ID: 198**

Message Type: WARNING

Message:

Length `propertyLength` of a property attached to a text record exceeds the limit `maximumLimit`. This property will be dropped.

**Message ID: 199**

Message Type: WARNING

Message:

Length `propertyLength` of a property attached to a `SREF` record exceeds the limit `maximumLimit`. This property will be dropped.

**Message ID: 200**

Message Type: WARNING

Message:
Length $propertyLength$ of a property attached to a $AREF$ record exceeds the limit $maximumLimit$. This property will be dropped.

**Message ID: 201**

Message Type: WARNING

Message:

Failed to get a string corresponding to the boundary property value from the Stream record type $STRM_PROPVALUE (44)$.

Extended Help: PIPO failed to get a string corresponding to the boundary property value from the record type $STRM_PROPVALUE (44)$ because the record is corrupt. The $BOUNDARY$ construct containing this record will not be translated.

**Message ID: 202**

Message Type: WARNING

Message:

Failed to get a string corresponding to the path property value from the Stream record type $STRM_PROPVALUE (44)$.

Extended Help: PIPO failed to get a string corresponding to the path property value from the record type $STRM_PROPVALUE (44)$ because the record is corrupt. The $PATH$ construct containing this record will not be translated.

**Message ID: 203**

Message Type: WARNING

Message:

Failed to get a string corresponding to the box property value from the Stream record type $STRM_PROPVALUE (44)$.

Extended Help: PIPO failed to get a string corresponding to the box property value from the record type $STRM_PROPVALUE (44)$ because the record is corrupt. The $BOX$ construct containing this record will not be translated.
Message ID: 204

Message Type: WARNING

Message:

Failed to get a string corresponding to the text property value from Stream record type STRM_PROPVALUE (44).

Extended Help: PIPO failed to get a string corresponding to the text property value from the record type STRM_PROPVALUE (44) because the record is corrupt. The TEXT construct containing this record will not be translated.

Message ID: 205

Message Type: WARNING

Message:

The SKILL function piUserShapeHandler() modified the boundary at layer layerNumber, startpoint xcoord, ycoord.

Extended Help: If you define a SKILL function piUserShapeHandler() before running Stream In, all the shapes defined in Stream file will pass through this function before getting translated. The user-defined SKILL function might modify or drop the shape.

Message ID: 206

Message Type: WARNING

Message:

The SKILL function piUserShapeHandler() did not set the variable pipoArray while processing boundary at layer layerNumber, startpoint xcoord, ycoord.

Extended Help: When you define the SKILL function piUserShapeHandler() before running Stream In, all the shapes defined in Stream file will pass through this function before getting translated. The user defined SKILL function might modify or drop the shape.

Message ID: 207

Message Type: WARNING
Message:

The SKILL function `piUserShapeHandler()` dropped the boundary at layer `layerNumber`, startpoint `xcoord, ycoord`.

Extended Help: When you define the SKILL function `piUserShapeHandler()` before running Stream In, all the shapes defined in Stream file will pass through this function before getting translated. The user defined SKILL function might modify or drop the shape.

**Message ID: 208**

Message Type: WARNING

Message:

The SKILL function `piUserShapeHandler()` modified the path at layer `layerNumber`, startpoint `xcoord, ycoord`.

Extended Help: When you define the SKILL function `piUserShapeHandler()` before running Stream In, all the shapes defined in Stream file will pass through this function before getting translated. The user-defined SKILL function might modify or drop the shape.

**Message ID: 209**

Message Type: WARNING

Message:

The SKILL function `piUserShapeHandler()` could not set the variable `pipoArray` while processing path at layer `layerNumber`, startpoint `xcoord, ycoord`.

Extended Help: When you define the SKILL function `piUserShapeHandler()` before running Stream In, all the shapes defined in Stream file will pass through this function before getting translated. The user defined SKILL function might modify or drop the shape.

**Message ID: 210**

Message Type: WARNING

Message:

The SKILL function `piUserShapeHandler()` dropped the path at layer `layerNumber`, startpoint `xcoord, ycoord`. 
Extended Help: If you define a SKILL function `piUserShapeHandler()` before running Stream In, all the shapes defined in Stream file will pass through this function before getting translated. The user defined SKILL function might modify or drop the shape.

**Message ID: 211**

Message Type: WARNING

Message:

Unexpected Stream record type `streamRecord` encountered in the boundary record.

Extended Help: The input Stream file contains invalid record sequence. Only the following record types can occur between the Stream record type `STRM_BOUNDARY` (8) and `STRM_ENDEL` (17)- `STRM_LAYER` (13), `STRM_DATATYPE` (14), `STRM_XY` (16), `STRM_PROPATTR` (43), `STRM_PROPVALUE` (44).

**Message ID: 212**

Message Type: WARNING

Message:

Failed to get an integer corresponding to the pathtype from the Stream record type `STRM_PATHTYPE` (33).

Extended Help: PIPO failed to get an integer corresponding to `PATHTYPE` datatype from the Stream record type `STRM_PATHTYPE` (33) because the record is corrupt. The `PATH` construct containing this record will not be translated.

**Message ID: 213**

Message Type: WARNING

Message:

Invalid pathtype `pathType` encountered in the Stream file. This path will be dropped.

Extended Help: The input Stream file contains an invalid pathtype in the path record. Path types supported by Stream format are 0,1,2,4.
Message ID: 214
Message Type: WARNING
Message:
Failed to get an integer corresponding to the path width from the Stream record type STRM_WIDTH (15).

Extended Help: PIPO failed to get an integer corresponding to PATH width from the record type STRM_WIDTH (15) because the record is corrupt. The PATH construct containing this record will not be translated.

Message ID: 215
Message Type: WARNING
Message:
Failed to get an integer value corresponding to the begin extension of the path from the Stream record type STRM_BGNEXTN (48).

Extended Help: PIPO failed to get an integer corresponding to path begin extension from the Stream record type STRM_BGNEXTN (48) because the record is corrupt. The PATH construct containing this record will not be translated.

Message ID: 216
Message Type: WARNING
Message:
The number of points numberOfPoints in a path is out of the range 2-maximumRange supported by the Cadence database. Path will be dropped.

Message ID: 217
Message Type: INFORMATIONAL
Message:
The number of points numberOfPoints in a path exceed maximum number maximumPoints supported in Stream format.
Message ID: 218

Message Type: WARNING

Message:

Unexpected Stream record type `streamRecordType` encountered in the path record.

Extended Help: The input Stream file contains invalid record sequence. Only the following record types can occur between the Stream record type `STRM_PATH` (9) and `STRM_ENDEL` (17) - `STRM_LAYER` (13), `STRM_DATATYPE` (14), `STRM_XY` (16), `STRM_PROPATTR` (43), `STRM_PROPVALUE` (44), `STRM_ENDEXTN` (49), `STRM_BGNEXTN` (48), `STRM_WIDTH` (15), `STRM_PATHTYPE` (33).

Message ID: 219

Message Type: FATAL

Message:

Unexpected EOF encountered in a path record in the input Stream file.

Extended Help: The input Stream file contains invalid record sequence. A path record should begin with the Stream record type `STRM_PATH` (9) and should end with `STRM_ENDEL` (17). In a path record `STRM_ENDEL` (17) was not encountered.

Message ID: 220

Message Type: WARNING

Message:

Invalid boxtype `boxType` encountered in the Stream file. Valid range is 0-255.

Message ID: 221

Message Type: WARNING

Message:

The first `xcoord1, ycoord1` and the last `xcoord2, ycoord2` vertexes of a box are not same. Box will be dropped.
Message ID: 222

Message Type: FATAL

Message:

Unexpected EOF encountered in box record in the input Stream file.

Extended Help: The input Stream file contains invalid record sequence. A box record should begin with the Stream record type STRM_PATH (9) and should end with STRM_ENDEL (17).

Message ID: 223

Message Type: WARNING

Message:

Invalid nodetype nodeType encountered in the Stream file. Valid range is '0-255'.

Message ID: 224

Message Type: WARNING

Message:

The number of vertexes totalVertexes in a node exceeds limit maximumLimit. Node will be dropped.

Message ID: 225

Message Type: WARNING

Message:

Unexpected Stream record type streamRecord encountered in the node record. Cell containing this record will be dropped.

Extended Help: The input Stream file contains invalid record sequence. Only the following record types can occur between the Stream record type STRM_NODE (21) and STRM_ENDEL (17) - STRM_LAYER (13), STRM_NODETYPE (42), STRM_XY (16), STRM_PROPATTR (43), STRM_PROPVALUE (44), STRM_ELFLAGS (38), and STRM_PLEX (47).
Message ID: 226
Message Type: FATAL
Message:

Unexpected EOF encountered in node record in the input Stream file. A node record should begin with the Stream record type STRM_NODE (21) and should end with STRM_ENDEL (17).

Message ID: 227
Message Type: WARNING
Message:

Failed to get a string corresponding to a structure name from the Stream record type STRM_SNAME (18).

Extended Help: PIPO failed to get a string corresponding to a structure name from the record type STRM_SNAME (18) as the record is corrupt. The SREF construct containing this record will not be translated.

Message ID: 228
Message Type: WARNING
Message:

Failed to get integer array values corresponding to the mirror information from the Stream record type STRM_STRANS (26).

Extended Help: PIPO failed to get a real number corresponding to the mirror information from the Stream record type STRM_STRANS (26) because the record is corrupt. The SREF construct containing this record will not be translated.

Message ID: 229
Message Type: WARNING
Message:

Failed to get a real number corresponding to the translation angle from the Stream record type STRM_ANGLE (28).
Extended Help: PIPO failed to get a real number corresponding to a translation angle from the Stream record type \texttt{STRM\_ANGLE (28)} because the record is corrupt. The \texttt{SREF} construct containing this record will not be translated.

**Message ID: 230**

Message Type: WARNING

Message:

Failed to get a real number corresponding to the \texttt{magnification} information from the Stream record type \texttt{STRM\_MAG (27)}.

Extended Help: PIPO failed to get a real number corresponding to the \texttt{magnification} information from the Stream record type \texttt{STRM\_MAG (27)} because the record is corrupt. The \texttt{SREF} construct containing this record will not be translated.

**Message ID: 231**

Message Type: WARNING

Message:

Failed to get an integer array corresponding to the coordinates of a structure from the Stream record type \texttt{STRM\_XY (16)}.

Extended Help: PIPO failed to get an integer array corresponding to the coordinates of a structure from the Stream record type \texttt{STRM\_XY (16)} because the record is corrupt. The \texttt{SREF} construct containing this record will not be translated.

**Message ID: 232**

Message Type: WARNING

Message:

Unexpected Stream record type \texttt{streamRecord} encountered in SREF record. The SREF record will not be translated.

Extended Help: The input Stream file contains an invalid record sequence. Only the following Stream record types can occur between the Stream record types \texttt{STRM\_SREF (10)} and \texttt{STRM\_ENDEL (17)} - \texttt{STRM\_LAYER (13)}, \texttt{STRM\_DATATYPE (14)}, \texttt{STRM\_XY (16)},
STRM_PROPATTR (43), STRM_PROPVALUE (44), STRM_ANGLE (28), STRM_MAG (27), STRM_STRANS (26), STRM_SNAME (18).

**Message ID: 233**

Message Type: WARNING

Message:

Unexpected **EOF** encountered in an **SREF** record of the input Stream file.

Extended Help: The input Stream file contains an invalid record sequence. An **SREF** record should begin with the Stream record type **STRM_SREF** (10) and should end with the Stream record type **STRM_ENDEL** (17).

**Message ID: 234**

Message Type: WARNING

Message:

Unexpected Stream record type **streamRecord** encountered in the **AREF** record.

Extended Help: The input Stream file contains an invalid record sequence. Only Stream record types greater than 15 can occur between the Stream record types **STRM_AREF** (11) and **STRM_ENDEL** (17).

**Message ID: 235**

Message Type: WARNING

Message:

Failed to get a string corresponding to an array instance name from the Stream record type **STRM_SNAME** (18).

Extended Help: PIPO failed to get a string corresponding to an array instance name from the Stream record type **STRM_SNAME** (18) because the record is corrupt. The **AREF** construct containing this record will not be translated.
Message ID: 236
Message Type: WARNING
Message:
Failed to get an integer corresponding to the mirror information from the Stream record type STRM_STRANS (26).

Extended Help: PIPO failed to get an integer array corresponding to the mirror information from the record type STRM_STRANS (26) because the record is corrupt. The AREF construct containing this record will not be translated.

Message ID: 237
Message Type: WARNING
Message:
Failed to get integer array corresponding to column/row specified in the Stream record type STRM_STRANS (26).

Extended Help: PIPO failed to get an integer array corresponding to the column/row specified in the Stream record type STRM_STRANS (26) because the record is corrupt. The AREF construct containing this record will not be translated.

Message ID: 238
Message Type: WARNING
Message:
Failed to get a real number corresponding to the translation angle information from the Stream record type STRM_ANGLE (28).

Extended Help: PIPO failed to get a real number corresponding to the translation angle information from the Stream record type STRM_ANGLE (28) because the record is corrupt. The AREF construct containing this record will not be translated.

Message ID: 239
Message Type: WARNING
Message:
Failed to get a real number corresponding to the magnification information from the Stream record type STRM_MAG (27).

Extended Help: PIPO failed to get a real number corresponding to the magnification information from the Stream record type STRM_MAG (27) because the record is corrupt. The AREF construct containing this record will not be translated.

Message ID: 240
Message Type: WARNING

Message:
Failed to get integer array corresponding to the coordinates of an array from the Stream record type STRM_XY (16).

Extended Help: PIPO failed to get an integer array corresponding to coordinates of a structure from the Stream record type STRM_XY (16) because the record is corrupt. The AREF construct containing this record will not be translated.

Message ID: 241
Message Type: WARNING

Message:
Unexpected Stream record type streamRecord encountered in AREF record.

Extended Help: The input Stream file contains an invalid record sequence. Only the following Stream record types can occur between the Stream record types STRM_SREF (10) and STRM_ENDEL (17)- STRM_LAYER (13), STRM_DATATYPE (14), STRM_XY (16), STRM_PROPATTR (43), STRM_PROPVALUE (44), STRM_ANGLE (28), STRM_MAG (27), STRM_STRANS (26), STRM_SNAME (18), and STRM_COLROW (19).

Message ID: 242
Message Type: FATAL

Message:
Unexpected EOF encountered in AREF record in the input Stream file.
Extended Help: The input Stream file contains an invalid record sequence. An AREF record can only begin with the Stream record type STRM_AREF (10) and needs to end with the Stream record type STRM_ENDEL (17).

Message ID: 243
Message Type: WARNING
Message:
Failed to get an integer corresponding to a text type from the Stream record type STRM_TEXTTYPE (22).

Extended Help: PIPO failed to get an integer corresponding to the text type from the record type STRM_TEXTTYPE (22) because the record is corrupt. The TEXT construct containing this record will not be translated.

Message ID: 244
Message Type: WARNING
Message:
Invalid text type textType encountered in Stream file. Valid range is 0-255.

Message ID: 245
Message Type: WARNING
Message:
Failed to get an integer corresponding to the presentation information from the Stream record type STRM_PRESENTATION (23).

Extended Help: PIPO failed to get an integer corresponding to the presentation information from the record type STRM_PRESENTATION (23) because the record is corrupt. The TEXT construct containing this record will not be translated.

Message ID: 246
Message Type: WARNING
Message:

Bad text justification $x_{\text{Justification}}, y_{\text{Justification}}$. The value of $x$ and $y$ justifications should not exceed two.

**Message ID: 247**

Message Type: WARNING

Message:

Failed to get an integer corresponding to the text path type from the Stream record type STRM_PATHTYPE (33).

Extended Help: PIPO failed to get an integer corresponding to text path type from the record type STRM_PATHTYPE (33) because the record is corrupt. The TEXT construct containing this record will not be translated.

**Message ID: 248**

Message Type: WARNING

Message:

Failed to get integer corresponding to the text width from the Stream record type STRM_WIDTH (26).

Extended Help: PIPO failed to get an integer corresponding to text width from the Stream record type STRM_WIDTH (26) because the record is corrupt. The TEXT construct containing this record will not be translated.

**Message ID: 249**

Message Type: WARNING

Message:

Failed to get integer array values corresponding to text mirror information from the Stream record type STRM_STRANS (26).

Extended Help: PIPO failed to get an integer array corresponding to text mirror information from the Stream record type STRM_STRANS (26) because the record is corrupt. The TEXT construct containing this record will not be translated.
Message ID: 250

Message Type: WARNING

Message:

Failed to get a real number value corresponding to text angle from the Stream record type STRM_ANGLE (28).

Extended Help: PIPO failed to get a real number corresponding to the text angle from the Stream record type STRM_ANGLE (28) because the record is corrupt. The TEXT construct containing this record will not be translated.

Message ID: 251

Message Type: WARNING

Message:

Failed to get a real number value corresponding to the text magnification information from the Stream record type STRM_MAG (27).

Extended Help: PIPO failed to get a real number corresponding to text magnification from the Stream record type STRM_MAG (27) because the record is corrupt. The SREF construct containing this record will not be translated.

Message ID: 252

Message Type: WARNING

Message:

Failed to get integer array values corresponding to the coordinates of a text from a Stream record type STRM_XY (16).

Extended Help: PIPO failed to get an integer array corresponding to the coordinates of a text from the Stream record type STRM_XY (16) because the record is corrupt. The TEXT construct containing this record will not be translated.

Message ID: 253

Message Type: WARNING
Message:

Failed to get a string value corresponding to text string from a Stream record type STRM_STRING (25).

Extended Help: PIPO failed to get a string corresponding to the text string from the record type STRM_STRING (25) because the record is corrupt. The TEXT construct containing this record will not be translated.

Message ID: 254
Message Type: WARNING

Message:

Unexpected Stream record type streamRecord encountered in the TEXT record.

Extended Help: The input Stream file contains an invalid record sequence. Only the following Stream record types can occur between the Stream record types STRM_TEXT (12) and STRM_ENDEL (17) - STRM_LAYER (13), STRM_DATATYPE (14), STRM_XY (16), STRM_PROPATTR (43), STRM_PROPVALUE (44), STRM_STRING (25), STRM_ANGLE (28), STRM_MAG (27), STRM_STRANS (26), STRM_WIDTH (15), and STRM_PATHTYPE (33).

Message ID: 255
Message Type: FATAL

Message:

Unexpected EOF encountered in the TEXT record of the input Stream file. A TEXT record can only begin with the Stream record type STRM_TEXT (12) and needs to end with STRM_ENDEL (17).

Message ID: 256
Message Type: WARNING

Message:

The Stream layer number layerNumber in the input stream file or layer map file is out of valid range 0-255.
Message ID: 257
Message Type: WARNING
Message:
The Stream layer layerNumber is mapped to an illegal dfll layer name in the layer map file. dfll layer name should be a string.

Message ID: 258
Message Type: WARNING
Message:
The Stream datatype dataType is mapped to an illegal dfll purpose in the layer map file.

Message ID: 259
Message Type: WARNING
Message:
Failed to open layer map file fileName. Check that the file exists in the run directory and that you have permission to read it.

Message ID: 260
Message Type: WARNING
Message:
Layer map file contains unknown purpose purposeName at line lineNumber. This entry in the layer map file will be ignored.

Extended Help: PIPO encountered a purpose in the layer map file, which is not defined in the technology file. This entry, in the Layer name mapping file, will be ignored.

Message ID: 261
Message Type: WARNING
Message:
Layer map file contains an unknown layer \textit{layerName} at line \textit{lineNumber}.

Extended Help: PIPO encountered a layer, in the layer map file, which is not defined in technology file. This entry, in the layer map file, will be ignored.

\textbf{Message ID: 262}

Message Type: WARNING

Message:

The purpose for layer \textit{layerName}, defined in layer map file, is not within the range 0-\textit{maximumRange}.

\textbf{Message ID: 263}

Message Type: WARNING

Message:

Layer map file contains multiple layer \textit{layerName} at line \textit{lineNumber}. Multiple layers are not allowed.

\textbf{Message ID: 264}

Message Type: WARNING

Message:

Less than three fields defined in the layer map file at line \textit{lineNumber}. This entry will be ignored.

\textbf{Message ID: 265}

Message Type: ERROR

Message:

Layer map file does not have a number in the third/fourth column at line \textit{lineNumber}.
Message ID: 266
Message Type: WARNING
Message:

The Stream layer number \textit{layerNumber} at line \textit{lineNumber}, in the layer map file, is not within the range 0-\textit{maximumRange}.

Extended Help: During Stream Out, entries containing invalid Stream layer or Stream datatype in the layer map file will be ignored. Due to this, all objects on the Cadence layer that is mapped to the invalid Stream layer or Stream datatype will also be ignored.

During Stream In, when a Stream layer or a Stream datatype greater than 7FFF is encountered in the layer map file, a warning is displayed but all the objects on this Stream layer or Stream datatype are translated. Stream layers or Stream datatypes greater than 7FFF are translated to support Stream files created by other vendors. However, the negative Stream layers or Stream datatypes are not supported and their objects will not be translated.

Message ID: 267
Message Type: WARNING
Message:

The Cadence design contains layer \textit{layerNumber} which is not defined in the technology file. This layer will be ignored.

Message ID: 268
Message Type: WARNING
Message:

Layer-purpose pair \textit{layerName}:\textit{purposeName} is not defined in the layer map file. This layer-purpose pair will be ignored.

Message ID: 269
Message Type: WARNING
Message:
Datatype *dataType* defined in layer map file at line *lineNumber* should be a number. PIPO will ignore this entry.

**Message ID: 270**

Message Type: WARNING

Message:

Datatype *dataType* at line *lineNumber*, in the layer map file, is not within range 0-*maximumRange*.

Extended Help: During Stream In, when a Stream layer or a Stream datatype greater than 255 is encountered in the layer map file, a warning is displayed but all the objects on this Stream layer or Stream datatype are translated. Stream layers or Stream datatypes greater than 255 are translated to support Stream files created by other vendors. However, the negative Stream layers or Stream datatypes are not supported and their objects will not be translated.

**Message ID: 271**

Message Type: WARNING

Message:

Failed to open font mapping file *fileName*. Check that the file exists in the run directory and that you have read permission.

**Message ID: 272**

Message Type: WARNING

Message:

Illegal Stream font *fontName* encountered, in font map file, at line *lineNumber*. Check that the Stream font is a number. Default value 0 will be used.

**Message ID: 273**

Message Type: WARNING

Message:
An illegal Cadence font, *fontName*, has been encountered at line *lineNumber* in the font map file. The default font *Stick* will be used.

Extended Help: Valid Cadence fonts you can use in a font map file are *euroStyle*, *gothic*, *math*, *roman*, *script*, *stick*, *fixed*, *swedish*, *raster*, and *milSpec*.

**Message ID: 274**
Message Type: WARNING
Message:

Less than two fields defined in the font map file at line *lineNumber*. This entry will be ignored.

**Message ID: 275**
Message Type: WARNING
Message:

Failed to open Property mapping file *fileName*. Check that file exists in the run directory and that you have read permission.

**Message ID: 276**
Message Type: WARNING
Message:

Attribute number *attributeNumber* at line *lineNumber*, in the Property mapping file, conflicts with the attribute number specified for *pinInfo*. This line will be ignored.

**Message ID: 277**
Message Type: WARNING
Message:

Illegal object type *objectType* encountered, in the Property mapping file, at line *lineNumber*.
Extended Help: Object types supported by property mapping file include rectangle, polygon, dot, line, path, arc label, ellipse, donut, instance, and mosaic. For more information on property mapping, refer to Cadence documentation.

**Message ID: 278**

Message Type: WARNING

Message:

Stream attribute number `attributeNumber`, defined in **Property mapping file**, at line `lineNumber` is not within range 0-127.

**Message ID: 279**

Message Type: WARNING

Message:

Less than three fields defined in the **Property mapping file** at line `lineNumber`. This entry will be ignored.

**Message ID: 280**

Message Type: ERROR

Message:

Number of properties defined for an object, in the **Property mapping file**, exceeds limit `max Limit`.

**Message ID: 281**

Message Type: WARNING

Message:

Failed to open **Cell name mapping file** `fileName`. Check that the file exists in the run directory and that you have read permission.
Message ID: 282
Message Type: WARNING
Message:
Less than three fields defined in the cell name map file at line lineNumber. This entry will be ignored.

Message ID: 283
Message Type: WARNING
Message:
Failed to open the pin text mapping file fileName. Check that the file exists in the run directory and that you have read permission.

Message ID: 284
Message Type: WARNING
Message:
Less than two fields defined in the pin text mapping file at line lineNumber. This entry will be ignored.

Message ID: 285
Message Type: FATAL
Message:
Data translation unsuccessful.

Message ID: 286
Message Type: FATAL
Message:
Failed to open the data dump file *fileName*. Check that you have permissions to create the data dump file in the run directory.

**Message ID: 287**

Message Type: INFORMATIONAL

Message:
Stream In will not create the output database because the `dataDump` option is used.

**Message ID: 288**

Message Type: INFORMATIONAL

Message:
Stream In will not create the output database because the `techfileGen` option is used.

**Message ID: 291**

Message Type: WARNING

Message:
Number of points *numberOfPoints* in the path exceeds maximum limit *maximumLimit*.

**Message ID: 292**

Message Type: WARNING

Message:
Coincident point *xcoord*, *ycoord* encountered in the path. This point will be removed during conversion.

**Message ID: 293**

Message Type: WARNING

Message:
Bad path encountered. Number of points in path less than two.

**Message ID: 294**

Message Type: WARNING

Message:

Collinear point $x_{coord}, y_{coord}$ encountered in path. This point will be removed during conversion.

**Message ID: 295**

Message Type: WARNING

Message:

Collinear point $x_{coord}, y_{coord}$ encountered in polygon. This point will be removed during conversion.

**Message ID: 296**

Message Type: WARNING

Message:

Open polygon encountered at $x_{coord}, y_{coord}$.

**Message ID: 297**

Message Type: WARNING

Message:

Data precision lost by scaling. Initial value $number1$, scaled value $number2$.

**Message ID: 298**

Message Type: WARNING

Message:
Failed to chop polygon. Ignoring polygon.

Extended Help: PIPO chops a polygon into smaller polygons when the number of vertexes in the polygon exceeds the maximum limit specified by the Maximum Vertices in Path/Polygon option.

**Message ID: 299**

Message Type: WARNING

Message:

Stream In is unable to find the structure definition for the primary cell in the stream file. Ensure that you have specified the correct values with the Primary Cell and Input Stream File options.

**Message ID: 300**

Message Type: FATAL/ERROR

Message:

Bad design hierarchy encountered. Recursive cell reference is not supported by PIPO. Check the input design.

**Message ID: 301**

Message Type: WARNING

Message: Ignoring the Hierarchy Depth Limit option because the top cell name is not specified.

Extended Help: During Stream Out, the Hierarchy Depth Limit option is valid only if you specify a top cell name. The Hierarchy Depth Limit option is ignored if no top cell name is specified in the GUI or the user template file. The hierarchy depth is not considered for a library.

**Message ID: 302**

Message Type: WARNING

Message:
Duplicate cell name found in design. Renaming cellName1 to cellName2.

Extended Help: In a Stream file, each structure needs to have a unique name. During Stream Out if multiple cells with the same name are found, in different reference libraries, cell names are changed.

**Message ID: 303**

Message Type: WARNING

Message:

Zero-Area rectangle encountered at layer layerNumber and bounding box (xcoord ycoord) (xcoord ycoord). Rectangle will be dropped.

**Message ID: 304**

Message Type: WARNING

Message:

PIPO failed to convert point list, assigned by pipoErrShapesHandler(), to a bounding box. Rectangle will be dropped.

**Message ID: 305**

Message Type: WARNING

Message:

Number of points numberOfPoints existing in the node exceeds limit maxPoints. Node will be ignored.

**Message ID: 306**

Message Type: WARNING

Message:

Zero area polygon encountered at layer layerNumber, startpoint xcoord, ycoord. Polygon will be dropped.
Message ID: 307
Message Type: WARNING
Message:
Zero area polygon returned by pipoErrShapesHandler(). Polygon will be dropped.

Message ID: 308
Message Type: WARNING
Message:
Reentrant polygon encountered at layer layerNumber and startpoint xcoord, ycoord.

Message ID: 309
Message Type: WARNING
Message:
Non-45 degree angle polygon encountered at layer layerNumber and startpoint xcoord, ycoord.

Message ID: 310
Message Type: INFORMATIONAL
Message:
Number of vertexes numVertexes in polygon at layer layerNumber and startpoint xcoord, ycoord exceeds limit defined by maximumVertexes. PIPO will break this polygon into smaller polygons.

Message ID: 311
Message Type: WARNING
Message:
Path with negative width \textit{pathWidth} encountered at layer \textit{layerNumber} and startpoint \textit{xcoord, ycoord}. Absolute value of width will be used.

**Message ID: 312**

Message Type: WARNING

Message:

Single point path encountered at layer \textit{layerNumber} and startpoint \textit{xcoord, ycoord}. Path will be dropped.

**Message ID: 313**

Message Type: WARNING

Message:

Number of vertexes \textit{numVertexes} in path at layer \textit{layerNumber} and startpoint \textit{xcoord, ycoord} exceeds limit defined by \textit{maxVertexes}. PIPO will break this path into smaller paths.

**Message ID: 314**

Message Type: WARNING

Message:

Failed to break path at layer \textit{layerNumber} and startpoint \textit{xcoord, ycoord}. Number of vertexes exceeds limit defined by \textit{maxVertexes}. PIPO will translate the path as it is.

**Message ID: 315**

Message Type: WARNING

Message:

ROD directory used during Stream In is not the same as directory defined in input Stream file. Some information might get lost.
Message ID: 316
Message Type: WARNING
Message:
Length $propertyLength$ of property, defining pin connectivity, exceeds limit $maximumLimit$. Ignoring pin connectivity information.

Message ID: 317
Message Type: WARNING
Message:
Length of properties for an object exceeds limit $maximumLimit$. Property $propertyName$ will be dropped.

Extended Help: When you use the User-Defined Property Mapping File option to preserve object properties, the total length of properties combined should not exceed 128 if the Respect GDSII limits option is NOT used and 1024 if the Respect GDSII limits option is used.

Message ID: 318
Message Type: FATAL
Message:
Length of name for instance $instanceName$ exceeds limit $maximumLimit$.

Message ID: 319
Message Type: INFORMATIONAL
Message:
Magnification factor $magnificationFactor$ applied to an instance of the master cell $cellName$ at $xcoord$, $ycoord$. 
Message ID: 320
Message Type: WARNING
Message:
Magnification factor $magnificationFactor$ applied to an array instance of the master cell $cellName$ at $xcoord$, $ycoord$.

Message ID: 321
Message Type: WARNING
Message:
Null string found in the design at layer $layerName$, point $xcoord$, $ycoord$. String will be dropped.

Message ID: 322
Message Type: WARNING
Message:
Failed to create the $fileName$ file in the run directory. Check that you have permissions to create this file.

Message ID: 323
Message Type: WARNING
Message:
Error in the definition of $poCellNameMap()$. Cell name $cellName$ mapped to NULL.

Message ID: 324
Message Type: WARNING
Message:
Error in the definition of piLayerMap(). No layer string returned for layer layerNumber, datatype dataType.

**Message ID: 325**

**Message Type: WARNING**

**Message:**

Error in definition of piLayerMap(). No purpose string returned for layer layerNumber, datatype dataType.

**Message ID: 326**

**Message Type: WARNING**

**Message:**

Error in definition of piLayerMap(). Purpose purposeName returned by piLayerMap() is not defined in the technology file.

**Message ID: 327**

**Message Type: WARNING**

**Message:**

Layer layerNumber corresponding to layer layerName, returned by piLayerMap(), exceeds limit 255.

Extended Help: Maximum limit for layer numbers is 255. Either there are too many layers in the input Stream file being translated or the technology file definition for a layer has a layer number greater than 255.

**Message ID: 328**

**Message Type: WARNING**

**Message:**

Failed to get layer name for layer layerNumber from technology file fileName. All objects on the layer will be dropped.
Message ID: 329
Message Type: WARNING
Message:
Failed to get a purpose name for the purpose `purposeNumber` from the technology file `fileName`. All objects on the layer will be dropped.

Message ID: 330
Message Type: WARNING
Message:
Error in the definition of `poLayerMap()`. Invalid layer number returned for layer `layerName`.

Message ID: 331
Message Type: WARNING
Message:
Error in the definition of `poLayerMap()`. Invalid datatype returned for layer-purpose pair `layerName:purposeName`.

Message ID: 332
Message Type: WARNING
Message:
Error in the definition of `poLayerMap()`. Invalid datatype returned. Integer expected.

Message ID: 333
Message Type: WARNING
Message:
Failed to get Stream layer rule for layer-purpose pair `layerName:purposeName` from technology file. All objects on the layer will be dropped.
Message ID: 334
Message Type: WARNING
Message:
Stream layer rule for layer-purpose pair \textit{layerName:purposeName} contains a non-integer \textit{StreamNumber/Datatype}. All objects on the layer will be dropped.

Message ID: 335
Message Type: WARNING
Message:
Stream layer rule for layer-purpose pair \textit{layerName:purposeName} is \textit{nil}. All objects on the layer will be dropped.

Message ID: 336
Message Type: WARNING
Message:
Error in the definition of \textit{textFontMap()}. Expected an integer or list of two integers while mapping fonts for \textit{fontType}.

Message ID: 337
Message Type: WARNING
Message:
Number of purposes \textit{numPurposes} defined for layer \textit{layerName} exceeds limit \textit{maximumLimit}. Resetting value of purpose to 0.

Message ID: 338
Message Type: WARNING
Message:
Error in the definition of `pioErrShapesHandler()`. Invalid datatype returned while processing shape `shapeName`. Integer datatype expected.

**Message ID: 339**

Message Type: WARNING

Message:

Error in the definition of `pioErrShapesHandler()`. PIPO is unable to read the list `pioArray` while processing the shape `shapeName`.

**Message ID: 340**

Message Type: WARNING

Message:

Error in the definition of `pioErrShapesHandler()`. Invalid code returned while processing the shape `shapeName`.

**Message ID: 341**

Message Type: WARNING

Message:

Negative angle `angle` encountered for an instance of `masterName` at origin `xcoord`, `ycoord` in the Stream file. This will be converted to the positive angle `angle`.

Extended Help: Cadence database does not support negative angles. So, PIPO converts a negative angle to a positive angle by adding `n*360` to it. The value `n` is calculated as `(negative_angle /(-360))+1`). Therefore, negative angles between 0 and -360 are converted by adding 360. Negative angles from -360 to -720 are converted to positive angles by adding 2*360.

**Message ID: 342**

Message Type: WARNING

Message:
Negative angle \(\text{angle}\) encountered for a mosaic instance of \(\text{masterName}\) at origin \(xcoord, ycoord\) in the Stream file. This will be converted to the positive angle \(\text{angle}\).

Extended Help: Cadence database does not support negative angles. So, PIPO converts a negative angle to a positive angle by adding \(n \times 360\) to it. The value \(n\) is calculated as \((\text{negative\_angle} / (-360)) + 1\). Therefore, negative angles between 0 and -360 are converted by adding 360. Negative angles from -360 to -720 are converted to positive angles by adding 2\(\times 360\).

**Message ID: 343**

**Message Type: WARNING**

**Message:**

Negative angle \(\text{angle}\) encountered for the label \(\text{labelName}\) at origin \(xcoord, ycoord\) in the Stream file. This will be converted to the positive angle \(\text{angle}\).

Extended Help: Cadence database does not support negative angles. So, PIPO converts a negative angle to a positive angle by adding \(n \times 360\) to it. The value \(n\) is calculated as \((\text{negative\_angle} / (-360)) + 1\). Therefore, negative angles between 0 and -360 are converted by adding 360. Negative angles from -360 to -720 are converted to positive angles by adding 2\(\times 360\).

**Message ID: 344**

**Message Type: WARNING**

**Message:**

Illegal angle \(\text{angle1}\) encountered for the instance \(\text{instanceName}\) at origin \(xcoord, ycoord\) in the Stream file. Angle will be converted to \(\text{angle2}\).

**Message ID: 345**

**Message Type: WARNING**

**Message:**

Illegal angle \(\text{angle1}\) encountered for the mosaic instance \(\text{instanceName}\) at origin \(xcoord, ycoord\) in the Stream file. Angle will be converted to \(\text{angle2}\).
Message ID: 346
Message Type: WARNING
Message:

Illegal angle angle1 encountered for the label labelName at origin xcoord, ycoord in the Stream file. Angle will be converted to angle2.

Message ID: 347
Message Type: WARNING
Message:

Failed to get Stream layer rules for the layer-purpose pair layerNumber: purposeNumber. All objects on the layer will be dropped.

Message ID: 348
Message Type: WARNING
Message:

Failed to get the layer name for layer layerNumber from the technology file. All objects on the layer will be dropped.

Message ID: 349
Message Type: WARNING
Message:

Value of the optionName option, defined in template file, should be a string.

Message ID: 350
Message Type: WARNING
Message:
Invalid path generated using the `optionName` option and the run directory `directoryName`.

**Message ID: 351**

Message Type: WARNING

Message:

Value of the option `optionName`, defined in the template file, contains spaces. Spaces not allowed.

**Message ID: 352**

Message Type: WARNING

Message:

Value of the option `optionName`, defined in the template file, should be a float.

**Message ID: 353**

Message Type: WARNING

Message:

Value of the option `scale`, defined in the template file, is greater than one.

**Message ID: 354**

Message Type: WARNING

Message:

Option `techfile` is not defined in the template file. Default value `streamTF.il` will be used.

**Message ID: 355**

Message Type: WARNING

Message:
Property mapping file fileName, defined in the template file, does not exist in the run directory.

**Message ID: 356**
Message Type: WARNING
Message:
User SKILL file fileName, specified in the template file, does not exist in the run directory.

**Message ID: 357**
Message Type: WARNING
Message:
Error occurred while loading the SKILL file fileName.

**Message ID: 358**
Message Type: WARNING
Message:
Value of the pinTextMapTable option is not specified in the template file. Value required to convert pin to text.

**Message ID: 359**
Message Type: WARNING
Message:
Option dotToPolygon used in the template file is obsolete. Use convertDot instead.

**Message ID: 360**
Message Type: WARNING
Message:
Option `rectToBox` is not supported by GDSII version `gdsVersion`.

**Message ID: 361**
Message Type: WARNING
Message:
Failed to create the ROD directory `dirName` in the run directory. Check that you have permissions to create the ROD directory.

**Message ID: 362**
Message Type: WARNING
Message:
Property mapping file `fileName`, defined in the template file, does not exist in the run directory.

**Message ID: 363**
Message Type: WARNING
Message:
More than one primary cell defined in the template file. Only the first cell will be used.

**Message ID: 364**
Message Type: WARNING
Message:
Run directory `dirName` does not exist or is not a directory.

**Message ID: 365**
Message Type: WARNING
Message:
Failed to get a real number from a Stream record because the data length is negative.

Message ID: 366
Message Type: WARNING
Message:
Failed to get a real number from a Stream record because $dataType$ is an invalid datatype found in Stream record.

Extended Help: PIPO failed to get a real number from a Stream record because it does not have a valid real datatype. Valid datatypes for real numbers are: 4, 5.

Message ID: 367
Message Type: WARNING
Message:
Failed to get text from a Stream record because the text length is negative.

Message ID: 369
Message Type: FATAL
Message:
Encountered a bad Stream record in the Stream file. Either the stream file is corrupt or the gds file name has an incorrect compressed file extension.

Extended Help: PIPO supports the gzip, compress, and bzip2 formats. PIPO determines the compression format based on the extension of the Stream file name. The supported compressed file name extensions are .gz, .Z, .bz, and .bz2. PIPO uses gzip for .gz files, uncompress for .Z files, and bzip2 for .bz and .bz2 files.

Message ID: 370
Message Type: WARNING
Message:
A record with illegal datatype $dataType$ was found in the Stream file at record $recordNumber$.

Extended Help: Valid datatypes for a Stream record are: 0 (unsigned one-byte integer), 1 (two-byte integer), 2 (two-byte integer), 3 (two-byte integer), 4 (four-byte real), 5 (eight-byte real), 6 (text).

**Message ID: 371**

**Message Type: FATAL**

Message:

Unable to read a line from the cellview $libName/cellName/viewName$.

**Message ID: 372**

**Message Type: WARNING**

Message:

A two-point path at coordinates $xcoord, ycoord$, having the same start and end points. It will be dropped.

**Message ID: 373**

**Message Type: FATAL**

Message:

Unable to read a polygon converted from an ellipse from the cellview $libName/cellName/viewName$.

Extended Help: The ellipse are converted to polygons during translation to the Stream format.

**Message ID: 374**

**Message Type: FATAL**

Message:
Unable to read a polygon converted from a donut from the cellview \textit{libName/cellName/viewName}.

Extended Help: The donut are converted to polygons during translation to the Stream format.

**Message ID: 375**

Message Type: FATAL

Message:

Unable to read a line converted from an arc from the cellview \textit{libName/cellName/viewName}.

Extended Help: The arcs are converted to line during translation to the Stream format.

**Message ID: 376**

Message Type: WARNING

Message:

Unknown text associate type encountered while reading a text from the cellview \textit{libName/cellName/viewName}.

Extended Help: Stream Out encountered an unknown text associate type in the input database. Text associate types supported are \textit{dbcPropAssocType, dbcAttrAssocType}.

**Message ID: 377**

Message Type: FATAL

Message:

Unable to read a label from the cellview \textit{libName/cellName/viewName}.

**Message ID: 378**

Message Type: WARNING

Message:
Unknown path style encountered in the input database.

Extended Help: Stream Out encountered an unknown path style in the input database. Path styles supported are 
\texttt{dbcVarExtendExtend}, \texttt{dbcRoundRound}, \texttt{dbcExtendExtend}, and \texttt{dbcTruncateExtend}.

**Message ID: 379**

**Message Type:** FATAL

**Message:**

Failed to create an array instance because the master cell name \texttt{cellName} exceeds the maximum length \texttt{maximumLength} supported during Stream In. Use the cell name map file to map such cell names.

**Message ID: 380**

**Message Type:** WARNING

**Message:**

Failed to set the layer selectable for a newly created layer-purpose pair \texttt{layerName: purposeName}.

Extended Help: Stream In failed to set the layer selectable for a newly created layer-purpose pair. A new layer-purpose pair is created in the technology file when a layer-purpose pair used in Stream file is not defined in technology file, and you are not using the \texttt{Skip Undefined Layer-Purpose Pair} option.

**Message ID: 381**

**Message Type:** FATAL

**Message:**

Failed to find the cellview \texttt{cellName/viewName} in the target library during Stream In.

Extended Help: PIPO failed to find a cellview, defined in Stream file, in the target library after Stream In. This might be because of the reason that input Stream file contains the complete design and you have used the \texttt{Retain Reference Library (No Merge)} option.
Message ID: 382
Message Type: FATAL
Message:
Failed to open the cellview cellName/viewName from the target library in the append mode.

Message ID: 383
Message Type: WARNING
Message:
Failed to save or close the cellview cellName/viewName in the target library.

Message ID: 384
Message Type: WARNING
Message:
Too many reference libraries defined in the input Stream file. Maximum two reference libraries are allowed.

Message ID: 385
Message Type: WARNING
Message:
An array instance with an acute angle has been encountered in the input database. Array instance will be dropped.

Message ID: 386
Message Type: FATAL
Message:
Failed to get string corresponding to library name from the Stream record type STRM_LIBNAME (2).

Extended Help: PIPO failed to get a string corresponding to library name from the Stream record type STRM_LIBNAME (2) because the record is corrupt. The SREF construct containing this record will not be translated.

**Message ID: 387**

**Message Type: FATAL**

**Message:**

Unexpected **EOF** encountered in a structure record of the input Stream file. A structure record should begin with the Stream record type STRM_BGNSTR (5) and should end with the Stream record type STRM_ENDSTR (7).

**Message ID: 388**

**Message Type: WARNING**

**Message:**

Invalid datatype **dataType** encountered in the Stream file. Valid range is 0-255.

**Message ID: 389**

**Message Type: WARNING**

**Message:**

The number of vertexes **numVertices** in a boundary exceeds the maximum number of vertexes **maxVertices** supported by GDS format.

**Message ID: 390**

**Message Type: WARNING**

**Message:**

Unexpected **EOF** encountered in boundary record in the input Stream file.
Extended Help: The input Stream file contains invalid record sequence. A box record should begin with the Stream record type `STRM_BOUNDARY` (8) and should end with `STRM_ENDEL` (17).

**Message ID: 400**

Message Type: WARNING

Message:

Failed to get an integer value corresponding to the end extension of the path from the Stream record type `STRM_ENDEXTN` (49).

Extended Help: PIPO failed to get an integer corresponding to the path-end extension from the Stream record type `STRM_ENDEXTN` (49) because the record is corrupt. The `PATH` construct containing this record will not be translated.

**Message ID: 401**

Message Type: WARNING

Message:

Failed to get a string corresponding to the instance property value from the Stream record type `STRM_PROPVALUE` (44).

Extended Help: PIPO failed to get a string corresponding to the instance property value from the record type `STRM_PROPVALUE` (44) because the record is corrupt. The `NODE` construct containing this record will not be translated.

**Message ID: 402**

Message Type: WARNING

Message:

Failed to get a string corresponding to an array instance property value from the Stream record type `STRM_PROPVALUE` (44).

Extended Help: PIPO failed to get a string corresponding to the array instance property value from the record type `STRM_PROPVALUE` (44) because the record is corrupt. The `AREF` construct containing this record will not be translated.
**Message ID: 403**

Message Type: WARNING

Message:

The number of vertexes in a box are not five. Box will be dropped.

**Message ID: 404**

Message Type: WARNING

Message:

Failed to get a string corresponding to the node property value from the Stream record type STRM_PROPVALUE (44).

Extended Help: PIPO failed to get a string corresponding to the node property value from the record type STRM_PROPVALUE (44) because the record is corrupt. The NODE construct containing this record will not be translated.

**Message ID: 405**

Message Type: WARNING

Message:

Lattice format for an array in the input stream file is inconsistent with ANGLE/REFLECTION.

**Message ID: 406**

Message Type: WARNING

Message:

The layer map file contains an illegal Stream layer number layerNumber at line lineNumber.

**Message ID: 407**

Message Type: WARNING
Message:

Layer number \textit{layerNumber} at line \textit{lineNumber}, in the layer map file, is not within range 0-\textit{maximumLimit}.

**Message ID: 408**

Message Type: WARNING

Message:

Stream layer-datatype pair \textit{layerNumber}:\textit{purposeNumber} is not defined in the layer map file. This layer-purpose pair will be ignored.

**Message ID: 409**

Message Type: WARNING

Message:

Datatype \textit{dataType} at line \textit{lineNumber}, in the layer map file, is not within range 0-\textit{maximumRange}.

**Message ID: 410**

Message Type: WARNING

Message:

The layer \textit{layerNumber} defined in the pin text map file at line \textit{lineNumber} should be a number. PIPO will ignore this entry.

**Message ID: 411**

Message Type: WARNING

Message:

Coincident point \textit{xcoord}, \textit{ycoord} encountered in the polygon. This point will be removed during conversion.
**Message ID: 412**
Message Type: FATAL
Message:
Failed to create log file *filename* in the run directory. Check that you have permission to create this file.

**Message ID: 413**
Message Type: WARNING
Message:
Duplicate cellview found in design. The cellview *libName1/cellName1/viewName1* will be retained while the cellview *libName2/cellName2/viewName2* will be ignored.

**Message ID: 415**
Message Type: WARNING
Message:
Duplicate cell name found in design. Renaming *cellName1* to *cellName2*.
Extended Help: During Stream Out when multiple cells with the same name are found, in different reference libraries, cell names are changed. In a Stream file, each structure should have a unique name.

**Message ID: 416**
Message Type: WARNING
Message:
Acute angle polygon encountered at layer *layerNumber* and startpoint *xcoord, ycoord*.

**Message ID: 417**
Message Type: WARNING
Message:
Error in the definition of piCellNameMap(). Cell name cellName mapped to NULL.

**Message ID: 418**
Message Type: WARNING
Message:

Error in definition of poParamCellNameMap(). Cell name cellName mapped to NULL.

**Message ID: 419**
Message Type: WARNING
Message:

A NULL layer layerNumber has been specified in the layer mapping file. This layer will be dropped.

**Message ID: 420**
Message Type: WARNING
Message:

The SKILL list listName, is not defined in the template file.

**Message ID: 421**
Message Type: WARNING
Message:

The SKILL variable variableName is not assigned in the template file.

**Message ID: 422**
Message Type: WARNING
Message:
Value of the option \textit{optionName}, defined in the template file, should be an integer.

**Message ID: 423**

Message Type: ERROR

Message:

Failed to open the cross-referenced cellview \textit{libName/cellName/viewName}. Check that the library exists at the location defined in the \textit{cds.lib} file, and that the cellview exists.

**Message ID: 424**

Message Type: ERROR

Message:

Internal Error: Failed to generate ROD SKILL file names for the cellview \textit{libName/cellName/viewName} due to NULL library name.

**Message ID: 425**

Message Type: ERROR

Message:

Internal Error: Failed to generate ROD SKILL file names for the cellview \textit{libName/cellName/viewName} due to NULL cell name.

**Message ID: 426**

Message Type: ERROR

Message:

Internal Error: Failed to generate ROD SKILL file names for the cellview \textit{libName/cellName/viewName} due to NULL view name.

**Message ID: 427**

Message Type: FATAL
Message:

Internal Error: pcell variant cellName/viewName was not flattened.

**Message ID: 428**
Message Type: FATAL
Message:

Internal Error: Bad pass number passNumber due to memory corruption.

**Message ID: 429**
Message Type: FATAL
Message:

Internal Error: Failed to find cellview cellName/viewName in the cell record table.

**Message ID: 430**
Message Type: FATAL
Message:

Internal Error: Memory corruption occurred.

**Message ID: 431**
Message Type: FATAL
Message:

Unknown translator translatorName specified as argument to the pipo command.

**Message ID: 432**
Message Type: FATAL
Message:
Insufficient number of arguments specified.

**Message ID: 433**

Message Type: FATAL

Message:

Setup file *fileName* does not exist in the current working directory.

**Message ID: 434**

Message Type: FATAL

Message:

Failed to load the setup file *fileName* from the current working directory.

**Message ID: 435**

Message Type: WARNING

Message:

The `skipInstBinding` option is chosen. The extra processing done for binding instances in the top cellview will be skipped during Stream In.

**Message ID: 436**

Message Type: ERROR

Message:

An instance of *cellName* has magnification %4.2f. It will be dropped because OpenAccess does not support instance magnification.

**Message ID: 437**

Message Type: ERROR

Message:
An arrayed instance of cellName has magnification %4.2f. It will be dropped because OpenAccess does not support instance magnification.

**Message ID: 438**

Message Type: WARNING

Message:

The Techfile Choice option is being ignored because it is not supported in PIPO on OpenAccess.

Extended Help: Layer mapping rules can not be defined in the OpenAccess technology file because the streamLayers section is no longer supported in the OpenAccess technology file. Therefore, cellview level layer mapping is not possible using the Techfile Choice option.

**Message ID: 439**

Message Type: WARNING

Message:

Invalid associate of attribute type encountered while reading a text display object from the cellview libName/cellName/viewName.

Extended Help: Stream Out encountered an invalid associate of attribute type for a text display object in the input database. Text attribute associate types supported are dbcNetType, dbcTermType, dbcInstTermType, dbcPinType, dbcMosaicType, dbcInstType, dbcGroupType, and dbcPropType.

**Message ID: 440**

Message Type: FATAL

Message:

Stream In cannot search for decompresion_utility to uncompress the given stream file fileName because the environment variable PATH does not exist.

**Message ID: 441**

Message Type: FATAL
Message:
Stream In cannot uncompress the given file fileName because the executable executableName was not found in the path.

Message ID: 442
Message Type: WARNING

Message:
Stream Out cannot search for decompression_utility because the environment variable PATH does not exist. Stream Out will generate uncompressed Stream file fileName.

Message ID: 443
Message Type: WARNING

Message:
Stream Out will generate uncompressed Stream file fileName because the executable executableName was not found in the path.

Message ID: 444
Message Type: WARNING

Message:
Stream Out could not recognize the given compression_type compression. Valid compression values are gzip, bzip2, and compress.

Message ID: 445
Message Type: ERROR

Message:
User-defined SKILL function functionName returned a non-string value for TEXT textName. This TEXT will be dropped.
Design Data Translator's Reference
PIPO Messages

**Message ID: 446**

Message: WARNING

Message:

No mapping found for layer-purpose pair `layerName:purposeName`. All objects on this LPP will be dropped.

**Message ID: 447**

Message Type: WARNING

Message:

The value of `Hierarchy Depth Limit` option is `value`. All the cells beyond this hierarchy level are being dropped.

Extended Help: The input design contains more hierarchy levels than specified in the `Hierarchy Depth Limit` option. All the cells beyond this hierarchy level will not be translated. You can increase the value of `Hierarchy Depth Limit` option to translate the entire hierarchy.

**Message ID: 448**

Message Type: FATAL

Message:

Compressed files cannot be used with more than 32767 cells.

Extended Help: The input compressed Stream file contains more number of cells than that are allowed in a dfII library. You can decompress the Stream file and try Stream In.

**Message ID: 449**

Message Type: FATAL

Message:

Pcell evaluation failed. This could happen as a result of either a syntax error or the use of an unsupported PIPO function in Pcell SKILL code. Only db*() and rod*() functions can be used in Pcell SKILL code. To continue, either use the ‘Ignore Pcell evaluation failure’ option in Stream In/Out Options or fix the Pcell SKILL code.
Message ID: 450
Message Type: WARNING
Message:
SREF of master name `masterName` has more than one XY point.
Extended Help: According to the grammar of Stream format, SREF record cannot have more than one XY point.

Message ID: 451
Message Type: WARNING
Message:
`PROPATTR attributeNumber` was not in the valid range 1 to 127 in `recordName` record.
Extended Help: According to the grammar of Stream format value of property attribute should be in the range of 1 - 127.

Message ID: 452
Message Type: WARNING
Message:
`TEXT` has more than one co-ordinates.
Extended Help: According to the grammar of Stream format, `TEXT` record cannot have more than one co-ordinate.

Message ID: 453
Message Type: WARNING
Message:
The `PATHTYPE value` is not in the valid range 0 to 4.
Extended Help: According to the grammar of Stream format, `TEXT` record cannot have more than one co-ordinate.
**Message ID: 454**

Message Type: WARNING

Message:

Number of columns `numberofCols` in AREF of master name `masterName` are not in the valid range 0 to 32,767.

Extended Help: Number of columns cannot be greater than 32,767 in an AREF record.

---

**Message ID: 455**

Message Type: WARNING

Message:

Number of rows `numberofrows` in AREF of master name `masterName` are not in the valid range 0 to 32,767.

Extended Help: Number of rows cannot be greater than 32,767 in an AREF record.

---

**Message ID: 456**

Message Type: WARNING

Message:

A STRING of more than 512 characters is encountered in a text. It will be truncated to 512 chars and the new text string would be `string`.

Extended Help: According to the Stream format, String in a text can have at the most 512 characters.

---

**Message ID: 457**

Message Type: WARNING

Message:

AREF of master name `masterName` has more than three XY points.
Extended Help: According to the grammar of STREAM format, an AREF record cannot have more than three XY points.

**Message ID: 465**

Message Type: WARNING

Message:

Failed to update/create the cellview `libraryName/cellName/viewName`. Check that you have permissions to create or write into this cellview.

Extended Help: Stream In was unable to open the cellview in append mode in the target library. It might be because disk space is full or the number of subdirectories have exceeded the maximum limit.

**Message ID: 466**

Message Type: WARNING

Message:

The *Do Not Overwrite Existing Cell* and the *Append into existing database* options cannot be used together.

**Message ID: 467**

Message Type: WARNING

Message:

Data precision lost by snipping. Initial value `number1`, snapped to manufacturing grid as `number2`.

Extended Help: PIPO reports all precision loss if you set the value of `reportPrecision` to `t`. A precision loss happens when a coordinate of the geometry does not fit the manufacturing grid. For example, consider a library that has a manufacturing grid of .05 (50dbu) and there is a coordinate ‘12075’ in the geometry. When this coordinate is translated to a STREAM file, the coordinate is snapped to ‘12100’ and the above message is displayed.
Message ID: 468

Message Type: WARNING

Message:

Existing cells in the target library might be overwritten if the Stream file contains a cell with the same name as a cell in the target library. You can use the Do Not Overwrite Existing Cells option to avoid this problem.

Message ID: 469

Message Type: WARNING

Message:

Ignored the *ASCII Technology File Name* option because the target library already has an attached technology file.

Message ID: 470

Message Type: FATAL

Message:

Failed to open the pin text mapping file *fileName*. Check that the file exists in the run directory and that you have read permission on the file.

Message ID: 472

Message Type: WARNING

Message:

Pcell evaluation failed. This could have happened as a result of either a syntax error or the use of an unsupported PIPO function in Pcell SKILL code. Only db*() and rod*() functions can be used in Pcell SKILL code. The Pcell evaluation error in this case is ignored because the ‘Ignore Pcell evaluation failure’ option is selected in Stream In/Out Options.
CDB Abstract Cellview Properties for Exporting LEF/DEF

Introduction

There are two types of cellviews, layout and abstract, in a cell design. A layout cellview consists of the physical shapes of materials used on various routing layers. It also contains the geometrical information about pin shapes and cell shapes.

The layout cellview does not have any direct information about the nets (connectivity) because it consists of physical geometries only.

An abstract cellview is a high-level representation of a physical design. It represents data from masklayouts, layouts, or placed and routed autoLayouts. You use abstracts in bottom-up designs. An abstract view provides the boundary of a cell, pins, and the blockages information but no other physical information.

CDB abstract cellviews can be created in dflII by the abstract generator tool. They are also required to export LEF or DEF. To export LEF, CDB abstract cellviews require certain properties. The properties are maskLayoutSubType, prcellType, and prCellClass and some optional properties, such as function and placementClass.

This document provides information about the properties required by the abstract generator to export LEF and DEF to the dflII domain.

The maskLayoutSubType Property

While exporting LEF, the translator checks for the maskLayoutSubType property on the CDB abstract cellview in dflII. The values of this property must be equal to abstract.

Example

The following lines of code are required to define the maskLayoutSubType property with the value abstract:
The `prCellType` and `prCellClass` Property

The values of the `prCellType` and `prCellClass` properties depend on the type of classes in the macro statement. A macro is a keyword in LEF given to the cell in a design. The values for the `prCellType` property are `macro`, `IO`, and `standard`. The value for the `prCellClass` property are `cover`, `ring`, `block`, `input`, `output`, `InOut`, and `power`.

Consider the following syntax of the macro statement describing **Macro Cell Models** in a LEF file.

```plaintext
...[ CLASS { COVER | RING | BLOCK | PAD}
    [INPUT | OUTPUT | INOUT | POWER | SPACER]
    [ CORE
      [FEEDTHRU | TIEHIGH | TIELOW | SPACER | ANTENNACELL]
      [ENDCAP
        [PRE | POST | TOPLEFT | TOPRIGHT | BOTTOMLEFT
          | BOTTOMRIGHT}
    ] ; ]...
```

The different values that can be assumed by the `prCellType` and `prCellClass` properties depending upon the type of LEF statement are described below:

<table>
<thead>
<tr>
<th>LEF Statement</th>
<th>prCellType</th>
<th>prCellClass</th>
</tr>
</thead>
<tbody>
<tr>
<td>MACRO macroName CLASS</td>
<td>macro</td>
<td>cover</td>
</tr>
<tr>
<td>MACRO macroName CLASS</td>
<td>macro</td>
<td>ring</td>
</tr>
<tr>
<td>MACRO macroName CLASS</td>
<td>macro</td>
<td>block</td>
</tr>
<tr>
<td>MACRO macroName CLASS</td>
<td>IO</td>
<td></td>
</tr>
<tr>
<td>MACRO macroName CLASS</td>
<td>standard</td>
<td></td>
</tr>
<tr>
<td>MACRO macroName CLASS</td>
<td>IO</td>
<td>input</td>
</tr>
<tr>
<td>MACRO macroName CLASS</td>
<td>IO</td>
<td>output</td>
</tr>
<tr>
<td>MACRO macroName CLASS</td>
<td>IO</td>
<td>inOut</td>
</tr>
<tr>
<td>MACRO macroName CLASS</td>
<td>IO</td>
<td>power</td>
</tr>
<tr>
<td>MACRO macroName CLASS</td>
<td>IO</td>
<td>spacer</td>
</tr>
</tbody>
</table>
The function Property

To display the size of a macro while exporting LEF, the function property must exist on the layer-purpose pair of the abstract cellview that has a bounding box. This is required because while exporting LEF, the translator first checks for the function property on the specified layer-purpose pair of the cellview. Then, it returns the bounding box of the first figure on the layer-purpose pair as the size of the macro.

If the function property is not defined on any of the layer-purpose pairs, then it takes the bounding box of the abstract cellview as the size of the macro.

The value of the function property is cellBoundary.

The placementClass Property

The placementClass property defines all the valid placement locations and their specifications for the cells in a design. The value of the placementClass property is the
same as the name of the site defined in the SITE Section in LEF. This is an optional property and it is defined in the Site statement, which is optional in the macro statement.

**CDB Abstract Cellview Property for Exporting DEF**

To export DEF from a layout cellview, all the instances of the layout cellview must contain CDB abstract cellviews having the `prCellType` property. If this property is not present in the CDB abstract cellviews, following warning message is displayed:

*WARNING* No cell instances found in the design.

This message appears because the DEF writer is unable to find any acceptable component in the design. An acceptable component is any instance in the design that has a non-parameterized cellview and `prCellType` property value as macro, standard, IO, or corner.

In order to prevent this warning message, open the *Edit Cellview Properties* form (by pressing the Shift and q keys together) and add the `prCellType` property to the *Edit Cellview Properties* form. Set the value of the `prCellType` property to `standard` and then export DEF again.

Sometimes, even after adding this property, you might receive the following warning message:

*WARNING* Cellview (Inv layout) has no cell boundary.

This warning is displayed because the master cellview for the instance `Inv` does not have the value `prBoundary`. After adding `prBoundary`, DEF can be exported successfully.
CDL and SPICE

How CDL Differs from SPICE

CDL provides the following additional statements to the SPICE syntax:

- A `.global` statement declares power, ground, and clock. For each `.global` statement it encounters, *Import – CDL* defines a global node and the corresponding extra terminal for each cell, such as in the following example:

  `.global vdd1:P vdd2:P vss1:G vss2:G`

  To add the extra global I/O terminals to each subcircuit statement, follow the `.global` statement with a `.pin` statement.

- A slash (`/`) character in `.subckt` statements distinguishes input ports (following the slash) from output ports (preceding the slash). In this example, ports `a` and `b` are output ports, and ports `d` and `e` are input ports for cell `xxx`:

  `.subckt xxx a b / d e`

- A slash (`/`) character in element statements distinguishes a master cell name (following the slash) from the node names (preceding the slash). In this example, `q`, `r`, and `qb` are node names, and `NOR` is the master cell name:

  `xl q r qb /NOR`

- Allowance for empty cells (cells with only titles and I/O names given) for some applications. For example:

  `.subckt jkff q qn cl j k s r
  *... jkff is empty...
  .ends`
Sample SPICE File

The following is a SPICE file with all of the CDL extensions just discussed, except for an empty cell.

```
.global vdd:p vss:g
.pin vdd vss
.subckt nor out /a b
mp1 vdd a 1 vdd p 1=3u w=10u
mp2 1 b out vdd p 1=3u w=10u
mn1 out a vss vss n 1=3u w=5u
mn2 out b vss vss n 1=3u w=5u
.ends
.subckt rslatch q qb /r s
x1 q r qb /nor
x2 qb q s /nor
.ends
```
## Index

### Symbols

- ..., in syntax 15
- ... in syntax 15
- / in syntax 15
- [ ] in syntax 14
- {} in syntax 15
- | in syntax 14

### A

**ALIAS statements** 291

**Applicon, Export**

  - mapping 36
  - SKILL 279

  **mapping**
  - cell names 36
  - cell names with SKILL 280
  - layer names 37
  - pcell names 285

  **mapping tables, syntax rules** 36

  **SKILL**
  - pipoErrShapesHandler 283
  - poCellNameMap 280
  - poParamCellNameMap 285

  **translating**
  - illegal shapes 283

**Applicon, Import**

  - mapping 26
  - SKILL 271

  **mapping**
  - cell names 27
  - cell names with SKILL 271
  - fonts 32
  - fonts with SKILL 276
  - layer names 29
  - layer names with SKILL 272

  **mapping files, syntax rules** 26

  **SKILL**
  - pipoErrShapesHandler 277
  - textFontMap 276

  **translating**
  - illegal shapes 277

### B

- braces in syntax 15
- brackets in syntax 14

### C

**CDL format**

  - differences from Design Framework II 225
  - differences from SPICE 583

**CDL, Export**

  - blocking out cells 206
  - creating a black box 206
  - error and log messages 239

  **files**
  - .simrc 221
  - input and output 205
  - si.env 221
  - si.log 239
  - template file 211

  **translating**
  - global nodes 235
  - parameters 227
  - primitive devices 233

**CDL, Import**

  - error and log messages 203

  **files**
  - device-mapping 186
    - prepare 192
    - sample 195
    - syntax 192
    - input and output 182
    - ni.err 203
    - ni.log 203

  **template**
  - prepare 190
    - sample 190, 221

  **starting with nino** 189
  **starting with the menu command** 184, 209

  **translating**
  - global nodes 202
  - primitive devices 196

  **cell names, mapping**
command option
   CIF In   154
   CIF Out  169
   Stream In  63
   Stream Out  104
map table
   Applicon In  27
   Applicon Out  37
   CIF In  27
   CIF Out  37
   Stream In  27
   Stream Out  37
SKILL
   Export – CIF  280
   Import – CIF  271
   Stream, Export  280
   Stream, Import  271
changing grid resolution or design size  24
character mapping  292, 312
CIF format, differences with Design Framework II  160
CIF, Export
   ASCII dump file example  178
   error and log messages  176
   files
      input and output  164
      PIPO.LOG  162
      required  165
      template file  151
   form  166
   mapping
      cell names  27
      cell names with SKILL  271
      fonts  32
      fonts with SKILL  276
      layer names  29
      layer names with SKILL  273
   mapping files, syntax rules  26
   options form  154
   output options  153
   SKILL
      files  271
      piCifLayerMap  273
   starting with pipo  159
   starting with the menu command  158
   translating
      cell hierarchy  158
      data types  161
      illegal shapes  277
      paths  156
CIW  20
Command Interpreter Window. See CIW  20
commands
   CIF In  151
   CIF Out  166
   Import DEF command  310
   Physical menu
      CIF In  151
      CIF Out  166
      Stream In  57
      Stream Out  94
      Translators menu  21
   CONSTRAINTS statement, DEF data
      map  413
   conventions
      user-defined arguments  14
      user-entered text  14
   cutting polygons while translating  73, 116
D

data maps
  AntennaSize, LEF  385
  Arrays, LEF  370
  Constraints, DEF  413
  Design, DEF  388
  Groups, DEF  412
  IO Timings, DEF  399
  Macrocell Pin, LEF  376
  Pins, DEF  394
  Property Definitions, DEF  390
  regular wiring, DEF  405
  special wiring  409
  Units, LEF  324
  Vias, LEF  355
data types  431
  ASCII string  434
  bit array  432
  eight-byte real  433
  four-byte real  433
  four-byte signed integer  432
  two-byte signed integer  432
database units (Stream Out),
  interpreting  128
DEF
  data mapping. See also data
  maps  323, 387
  exporting from Preview  316
  DEF, importing to Preview  310
DEFINE statements  291
Design Framework II format, differences with
  CIF format  160
  Stream format  82, 127
DESIGN statement, DEF data map  388
device-mapping file  186
  prepare  192
  sample  195
  syntax  192

E

EDIF In  19
EDIF Out  19
example, Stream format  451

F

files
  .simrc  221
  Applicon, Export
    SKILL  279
  Applicon, Import
    SKILL  271
  CDL In device-mapping  186
    prepare  192
    sample  195
    syntax  192
  CDL In input and output  183
  CDL In template
    prepare  190
    sample  190, 221
  CDL Out template  211, 220
  CIF In input and output  150
  CIF In template  159
  CIF Out input and output  165
  CIF Out template  166
  CIF, Export SKILL  279
  CIF, Import SKILL  271
  ni.err, CDL In  203
  ni.log, CDL In  203
  PIPO.LOG
  CIF In  162
  CIF Out  176
    Stream In  85
    Stream Out  130
  si.log  239
  Stream In input and output  54 to 56
  Stream In template  54, 79
  Stream Out input and output  91 to 93
  Stream Out template  91, 96
  Stream, Export
    SKILL  279
  Stream, Import
    SKILL  270
fonts, mapping
  command option
    CIF In  155
    Stream In  101
    Stream Out  104
map table
  Applicon In  32
  CIF In  32
    Stream In  32
    SKILL
      Applicon, Import  276
Design Data Translator's Reference

CIF, Import 276
Stream, Export 282
Stream, Import 276
forms
CIF In 151
CIF In User-Defined Data 154
CIF Out 166
DEF In 311
LEF In 295
LEF Out 302
Stream In 57
Stream In Options 66
Stream In User-Defined Data 63
Stream Out 95
Write CellView to DEF File 317
function, undefined 45

K
keywords in syntax 14

L
labels, mapping. See pin text mapping 102
layer names, mapping
command option
CIF In 154
CIF Out 170
Stream In 64
Stream Out 103
map table
Applicon In 29
Applicon Out 37
CIF In 29
CIF Out 37
Stream In 29
Stream Out 37
SKILL
Applicon, Import 272
CIF, Export 281
CIF, Import 273
Stream, Import 272
Layers
Cut Layer 349
Masterslice Layer 350
Overlap Layer 351
Routing Layer 328
LEF
data mapping. See also data maps 323
exporting from Preview 302
importing to Preview 295
legal characters, in Preview and LEF/DEF 292
literal characters 14

M
mapping, cell names
command option
CIF In 154
CIF Out 169
Stream In 63
Stream Out 104
map table
Applicon In 27

GDS II format 430
GDS II. See Stream format 82
general translators 18
global nodes, translating
CDL In 202
CDL Out 235
grid resolution, changing 24
GROUPS statement, DEF data map 412

H
hierarchy, translating
CIF In 158
Stream In 72
Stream Out 115

I
illegal shapes, translating
Applicon, Export 283
Applicon, Import 277
CIF, Export 283
CIF, Import 277
Stream, Export 283
Stream, Import 277
Import DEF command 310
IOTIMINGS statement, DEF data map 399
italics in syntax 14
Design Data Translator's Reference

Applicon Out  37
CIF In 27
CIF Out 37
Stream In 27
Stream Out 37
SKILL 280
   Applicon, Import 271
   CIF, Export 280
   CIF, Import 271
   Stream, Export 280
   Stream, Import 271
mapping, fonts
   command option
      CIF In 155
      Stream In 64
      Stream Out 104
map table
   Applicon In 32
   CIF In 32
   Stream In 32
SKILL
   Applicon, Import 276
   CIF, Import 276
   Stream, Export 282
   Stream, Import 276
Stream equations 33
mapping, properties
   command option
      Stream In 64
      Stream Out 44
map table
   Stream In 35
   Stream Out 42
menu commands
   LEF and DEF overview 290
messages, error and log
   CDL In 203
   CDL Out 239
   CIF In 161
   CIF Out 176
   Stream In 84
   Stream Out 130
mustjoin pins 382
mustjoins, data equivalence 382

N
name collisions 292
netlist translators
   definition 18
   types of 19
ni.err file, CDL, Import 203
ni.log file, CDL, Import 203
nino, starting CDL, Import 189

O
obstructions, LEF data map 384
Or-bars in syntax 14

P
parameter inheritance 206, 227 to 230
parameterized cell names, mapping
   SKILL
      Applicon, Export 285
      Stream, Export 285
pcells
   safety rules for creating 45
   undefined function error 45
physical translators
   definition 18
   types of 19
### Design Data Translator's Reference

<table>
<thead>
<tr>
<th>piCellNameMap, Stream, Import</th>
<th>MessageID 130 498</th>
</tr>
</thead>
<tbody>
<tr>
<td>piCifLayerMap, CIF, Import</td>
<td>MessageID 131 498</td>
</tr>
<tr>
<td>piLayerMap, Stream, Import</td>
<td>MessageID 134 499</td>
</tr>
<tr>
<td>PIN statement, LEF data map</td>
<td>MessageID 136 499</td>
</tr>
<tr>
<td>pin text, mapping</td>
<td>MessageID 137 499</td>
</tr>
<tr>
<td>command option</td>
<td>MessageID 149 503</td>
</tr>
<tr>
<td>CIF Out 169</td>
<td>MessageID 15 469</td>
</tr>
<tr>
<td>map table</td>
<td>MessageID 151 504</td>
</tr>
<tr>
<td>Applicon Out 44</td>
<td>MessageID 152 504</td>
</tr>
<tr>
<td>CIF Out 44</td>
<td>MessageID 153 504</td>
</tr>
<tr>
<td>Stream Out 44</td>
<td>MessageID 155 504</td>
</tr>
<tr>
<td>pins</td>
<td>MessageID 156 505</td>
</tr>
<tr>
<td>in Preview 380</td>
<td>MessageID 160 505</td>
</tr>
<tr>
<td>map to cell terminals 380</td>
<td>MessageID 163 506</td>
</tr>
<tr>
<td>mustjoin 382</td>
<td>MessageID 164 506</td>
</tr>
<tr>
<td>strong-connect 380</td>
<td>MessageID 166 507</td>
</tr>
<tr>
<td>weak-connect 381</td>
<td>MessageID 17 470</td>
</tr>
<tr>
<td>PINS statement, DEF data map</td>
<td>MessageID 18 470</td>
</tr>
<tr>
<td>pipo</td>
<td>MessageID 219 522</td>
</tr>
<tr>
<td>starting CIF In 159</td>
<td>MessageID 22 471</td>
</tr>
<tr>
<td>starting CIF Out 174</td>
<td>MessageID 222 523</td>
</tr>
<tr>
<td>starting Stream In 77</td>
<td>MessageID 226 524</td>
</tr>
<tr>
<td>starting Stream Out 121</td>
<td>MessageID 23 471</td>
</tr>
<tr>
<td>PIPO Messages</td>
<td>MessageID 24 471</td>
</tr>
<tr>
<td>Error</td>
<td>MessageID 242 528</td>
</tr>
<tr>
<td>MessageID 10 468</td>
<td>MessageID 255 532</td>
</tr>
<tr>
<td>MessageID 265 534</td>
<td>MessageID 28 472</td>
</tr>
<tr>
<td>MessageID 280 538</td>
<td>MessageID 285 539</td>
</tr>
<tr>
<td>MessageID 300 542</td>
<td>MessageID 286 539</td>
</tr>
<tr>
<td>MessageID 423 568</td>
<td>MessageID 300 542</td>
</tr>
<tr>
<td>MessageID 424 568</td>
<td>MessageID 318 546</td>
</tr>
<tr>
<td>MessageID 425 568</td>
<td>MessageID 33 473</td>
</tr>
<tr>
<td>MessageID 426 568</td>
<td>MessageID 34 474</td>
</tr>
<tr>
<td>MessageID 436 570</td>
<td>MessageID 36 474</td>
</tr>
<tr>
<td>MessageID 437 570</td>
<td>MessageID 37 474</td>
</tr>
<tr>
<td>MessageID 446 572</td>
<td>MessageID 371 558</td>
</tr>
<tr>
<td>MessageID 54 479</td>
<td>MessageID 373 558</td>
</tr>
<tr>
<td>MessageID 70 488</td>
<td>MessageID 374 558</td>
</tr>
<tr>
<td>Fatal</td>
<td>MessageID 375 559</td>
</tr>
<tr>
<td>MessageID 1 466</td>
<td>MessageID 377 559</td>
</tr>
<tr>
<td>MessageID 10 468</td>
<td>MessageID 379 560</td>
</tr>
<tr>
<td>MessageID 101 490</td>
<td>MessageID 38 475</td>
</tr>
<tr>
<td>MessageID 106 491</td>
<td>MessageID 381 560</td>
</tr>
<tr>
<td>MessageID 107 491</td>
<td>MessageID 382 561</td>
</tr>
<tr>
<td>MessageID 108 491</td>
<td>MessageID 386 561</td>
</tr>
<tr>
<td>MessageID 11 468</td>
<td>MessageID 387 562</td>
</tr>
<tr>
<td>MessageID 110 492</td>
<td>MessageID 4 466</td>
</tr>
<tr>
<td>MessageID 115 493</td>
<td>MessageID 40 475</td>
</tr>
<tr>
<td>MessageID 12 468</td>
<td>MessageID 41 475</td>
</tr>
<tr>
<td>MessageID 125 496</td>
<td>MessageID 412 566</td>
</tr>
<tr>
<td>MessageID 13 469</td>
<td>MessageID 427 568</td>
</tr>
</tbody>
</table>
Design Data Translator's Reference

MessageID 428 569
MessageID 429 569
MessageID 430 569
MessageID 431 569
MessageID 432 569
MessageID 433 570
MessageID 434 570
MessageID 44 476
MessageID 440 571
MessageID 441 571
MessageID 448 573
MessageID 449 573
MessageID 45 477
MessageID 470 577
MessageID 48 477
MessageID 53 478
MessageID 55 479
MessageID 56 479
MessageID 58 479
MessageID 59 480
MessageID 6 467
MessageID 60 480
MessageID 62 481
MessageID 63 481
MessageID 64 481
MessageID 65 481
MessageID 66 481
MessageID 73 483
MessageID 76 484
MessageID 79 485
MessageID 80 485
MessageID 81 485
MessageID 85 486
MessageID 87 486
MessageID 90 487
MessageID 94 488
MessageID 105 490
MessageID 122 495
MessageID 161 506
MessageID 165 507
MessageID 167 507
MessageID 168 508
MessageID 169 508
MessageID 170 508
MessageID 171 508
MessageID 172 509
MessageID 173 509
MessageID 174 509
MessageID 175 510
MessageID 176 510
MessageID 177 510
MessageID 178 510
MessageID 16 469 470
MessageID 179 511  
MessageID 180 511  
MessageID 181 511  
MessageID 182 512  
MessageID 183 512  
MessageID 184 512  
MessageID 185 512  
MessageID 186 513  
MessageID 187 513  
MessageID 188 513  
MessageID 189 514  
MessageID 190 470  
MessageID 191 514  
MessageID 192 515  
MessageID 193 515  
MessageID 194 515  
MessageID 195 515  
MessageID 196 516  
MessageID 197 516  
MessageID 198 516  
MessageID 199 516  
MessageID 200 516  
MessageID 201 517  
MessageID 202 517  
MessageID 203 517  
MessageID 204 518  
MessageID 205 518  
MessageID 206 518  
MessageID 207 518  
MessageID 208 519  
MessageID 209 519  
MessageID 210 519  
MessageID 211 520  
MessageID 212 520  
MessageID 213 520  
MessageID 214 521  
MessageID 215 521  
MessageID 216 521  
MessageID 217 522  
MessageID 218 522  
MessageID 219 522  
MessageID 220 522  
MessageID 221 522  
MessageID 222 523  
MessageID 223 523  
MessageID 224 523  
MessageID 225 523  
MessageID 226 524  
MessageID 227 524  
MessageID 228 524  
MessageID 229 524  
MessageID 230 525  
MessageID 231 525  
MessageID 232 525  
MessageID 233 526  
MessageID 234 526  
MessageID 235 526  
MessageID 236 527  
MessageID 237 527  
MessageID 238 527  
MessageID 239 527  
MessageID 240 528  
MessageID 241 528  
MessageID 242 528  
MessageID 243 529  
MessageID 244 529  
MessageID 245 529  
MessageID 246 529  
MessageID 247 530  
MessageID 248 530  
MessageID 249 530  
MessageID 250 531  
MessageID 251 531  
MessageID 252 531  
MessageID 253 531  
MessageID 254 532  
MessageID 255 532  
MessageID 256 532  
MessageID 257 533  
MessageID 258 533  
MessageID 259 533  
MessageID 260 533  
MessageID 261 533  
MessageID 262 534  
MessageID 263 534  
MessageID 264 534  
MessageID 265 534  
MessageID 266 535  
MessageID 267 535  
MessageID 268 535  
MessageID 269 535  
MessageID 270 535  
MessageID 271 536  
MessageID 272 536  
MessageID 273 536  
MessageID 274 537  
MessageID 275 537  
MessageID 276 537  
MessageID 277 537  
MessageID 278 538  
MessageID 279 538  
MessageID 280 538  
MessageID 281 538  
MessageID 282 539  
MessageID 283 539  
MessageID 284 539
<table>
<thead>
<tr>
<th>MessageID</th>
<th>540</th>
<th>MessageID</th>
<th>551</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageID</td>
<td>291</td>
<td>MessageID</td>
<td>342</td>
</tr>
<tr>
<td>MessageID</td>
<td>292</td>
<td>MessageID</td>
<td>343</td>
</tr>
<tr>
<td>MessageID</td>
<td>293</td>
<td>MessageID</td>
<td>344</td>
</tr>
<tr>
<td>MessageID</td>
<td>294</td>
<td>MessageID</td>
<td>345</td>
</tr>
<tr>
<td>MessageID</td>
<td>295</td>
<td>MessageID</td>
<td>346</td>
</tr>
<tr>
<td>MessageID</td>
<td>296</td>
<td>MessageID</td>
<td>347</td>
</tr>
<tr>
<td>MessageID</td>
<td>297</td>
<td>MessageID</td>
<td>348</td>
</tr>
<tr>
<td>MessageID</td>
<td>298</td>
<td>MessageID</td>
<td>349</td>
</tr>
<tr>
<td>MessageID</td>
<td>299</td>
<td>MessageID</td>
<td>350</td>
</tr>
<tr>
<td>MessageID</td>
<td>3</td>
<td>MessageID</td>
<td>351</td>
</tr>
<tr>
<td>MessageID</td>
<td>30</td>
<td>MessageID</td>
<td>352</td>
</tr>
<tr>
<td>MessageID</td>
<td>301</td>
<td>MessageID</td>
<td>353</td>
</tr>
<tr>
<td>MessageID</td>
<td>302</td>
<td>MessageID</td>
<td>354</td>
</tr>
<tr>
<td>MessageID</td>
<td>303</td>
<td>MessageID</td>
<td>355</td>
</tr>
<tr>
<td>MessageID</td>
<td>304</td>
<td>MessageID</td>
<td>356</td>
</tr>
<tr>
<td>MessageID</td>
<td>305</td>
<td>MessageID</td>
<td>357</td>
</tr>
<tr>
<td>MessageID</td>
<td>306</td>
<td>MessageID</td>
<td>358</td>
</tr>
<tr>
<td>MessageID</td>
<td>307</td>
<td>MessageID</td>
<td>359</td>
</tr>
<tr>
<td>MessageID</td>
<td>308</td>
<td>MessageID</td>
<td>360</td>
</tr>
<tr>
<td>MessageID</td>
<td>309</td>
<td>MessageID</td>
<td>361</td>
</tr>
<tr>
<td>MessageID</td>
<td>31</td>
<td>MessageID</td>
<td>362</td>
</tr>
<tr>
<td>MessageID</td>
<td>311</td>
<td>MessageID</td>
<td>363</td>
</tr>
<tr>
<td>MessageID</td>
<td>312</td>
<td>MessageID</td>
<td>364</td>
</tr>
<tr>
<td>MessageID</td>
<td>313</td>
<td>MessageID</td>
<td>365</td>
</tr>
<tr>
<td>MessageID</td>
<td>314</td>
<td>MessageID</td>
<td>366</td>
</tr>
<tr>
<td>MessageID</td>
<td>315</td>
<td>MessageID</td>
<td>367</td>
</tr>
<tr>
<td>MessageID</td>
<td>316</td>
<td>MessageID</td>
<td>368</td>
</tr>
<tr>
<td>MessageID</td>
<td>317</td>
<td>MessageID</td>
<td>369</td>
</tr>
<tr>
<td>MessageID</td>
<td>318</td>
<td>MessageID</td>
<td>370</td>
</tr>
<tr>
<td>MessageID</td>
<td>319</td>
<td>MessageID</td>
<td>371</td>
</tr>
<tr>
<td>MessageID</td>
<td>320</td>
<td>MessageID</td>
<td>372</td>
</tr>
<tr>
<td>MessageID</td>
<td>321</td>
<td>MessageID</td>
<td>373</td>
</tr>
<tr>
<td>MessageID</td>
<td>322</td>
<td>MessageID</td>
<td>374</td>
</tr>
<tr>
<td>MessageID</td>
<td>323</td>
<td>MessageID</td>
<td>375</td>
</tr>
<tr>
<td>MessageID</td>
<td>324</td>
<td>MessageID</td>
<td>376</td>
</tr>
<tr>
<td>MessageID</td>
<td>325</td>
<td>MessageID</td>
<td>377</td>
</tr>
<tr>
<td>MessageID</td>
<td>326</td>
<td>MessageID</td>
<td>378</td>
</tr>
<tr>
<td>MessageID</td>
<td>327</td>
<td>MessageID</td>
<td>379</td>
</tr>
<tr>
<td>MessageID</td>
<td>328</td>
<td>MessageID</td>
<td>380</td>
</tr>
<tr>
<td>MessageID</td>
<td>329</td>
<td>MessageID</td>
<td>381</td>
</tr>
<tr>
<td>MessageID</td>
<td>330</td>
<td>MessageID</td>
<td>382</td>
</tr>
<tr>
<td>MessageID</td>
<td>331</td>
<td>MessageID</td>
<td>383</td>
</tr>
<tr>
<td>MessageID</td>
<td>332</td>
<td>MessageID</td>
<td>384</td>
</tr>
<tr>
<td>MessageID</td>
<td>333</td>
<td>MessageID</td>
<td>385</td>
</tr>
<tr>
<td>MessageID</td>
<td>334</td>
<td>MessageID</td>
<td>386</td>
</tr>
<tr>
<td>MessageID</td>
<td>335</td>
<td>MessageID</td>
<td>387</td>
</tr>
<tr>
<td>MessageID</td>
<td>336</td>
<td>MessageID</td>
<td>388</td>
</tr>
<tr>
<td>MessageID</td>
<td>337</td>
<td>MessageID</td>
<td>389</td>
</tr>
<tr>
<td>MessageID</td>
<td>338</td>
<td>MessageID</td>
<td>390</td>
</tr>
<tr>
<td>MessageID</td>
<td>339</td>
<td>MessageID</td>
<td>391</td>
</tr>
<tr>
<td>MessageID</td>
<td>340</td>
<td>MessageID</td>
<td>392</td>
</tr>
<tr>
<td>MessageID</td>
<td>341</td>
<td>MessageID</td>
<td>393</td>
</tr>
<tr>
<td>MessageID</td>
<td>342</td>
<td>MessageID</td>
<td>394</td>
</tr>
<tr>
<td>MessageID</td>
<td>566</td>
<td>MessageID</td>
<td>486</td>
</tr>
<tr>
<td>-----------</td>
<td>-----</td>
<td>-----------</td>
<td>-----</td>
</tr>
<tr>
<td>MessageID</td>
<td>416</td>
<td>MessageID</td>
<td>417</td>
</tr>
<tr>
<td>MessageID</td>
<td>418</td>
<td>MessageID</td>
<td>419</td>
</tr>
<tr>
<td>MessageID</td>
<td>42</td>
<td>MessageID</td>
<td>420</td>
</tr>
<tr>
<td>MessageID</td>
<td>421</td>
<td>MessageID</td>
<td>422</td>
</tr>
<tr>
<td>MessageID</td>
<td>43</td>
<td>MessageID</td>
<td>435</td>
</tr>
<tr>
<td>MessageID</td>
<td>438</td>
<td>MessageID</td>
<td>439</td>
</tr>
<tr>
<td>MessageID</td>
<td>442</td>
<td>MessageID</td>
<td>443</td>
</tr>
<tr>
<td>MessageID</td>
<td>446</td>
<td>MessageID</td>
<td>447</td>
</tr>
<tr>
<td>MessageID</td>
<td>450</td>
<td>MessageID</td>
<td>451</td>
</tr>
<tr>
<td>MessageID</td>
<td>454</td>
<td>MessageID</td>
<td>455</td>
</tr>
<tr>
<td>MessageID</td>
<td>458</td>
<td>MessageID</td>
<td>459</td>
</tr>
<tr>
<td>MessageID</td>
<td>462</td>
<td>MessageID</td>
<td>463</td>
</tr>
<tr>
<td>MessageID</td>
<td>466</td>
<td>MessageID</td>
<td>467</td>
</tr>
<tr>
<td>MessageID</td>
<td>470</td>
<td>MessageID</td>
<td>471</td>
</tr>
<tr>
<td>MessageID</td>
<td>474</td>
<td>MessageID</td>
<td>475</td>
</tr>
<tr>
<td>MessageID</td>
<td>479</td>
<td>MessageID</td>
<td>480</td>
</tr>
<tr>
<td>MessageID</td>
<td>483</td>
<td>MessageID</td>
<td>484</td>
</tr>
<tr>
<td>MessageID</td>
<td>488</td>
<td>MessageID</td>
<td>492</td>
</tr>
<tr>
<td>MessageID</td>
<td>496</td>
<td>MessageID</td>
<td>497</td>
</tr>
</tbody>
</table>

PIPO Performance Guidelines 139
PIPO-LOG file
CIF In 162
CIF Out 176
Stream In 85
Stream Out 130
pipoDisplay, Stream, Import 270
pipoErrShapesHandler
Applicon, Export 283
Applicon, Import 277
Stream, Export 283
Stream, Import 277
poCellNameMap
Applicon, Export 280
Stream, Export 280
poCifLayerMap, CIF, Export 281
polygons
cutting while translating 73, 116
reporting bad polygons
CIF In 156
CIF Out 171
Stream In 67
Stream Out 107
poParamCellNameMap
Applicon, Export 285
Stream, Export 285
PR Flatten
definition 19
primitive devices 196
syntax of instances 196
properties, mapping
command option
Stream In 64
map table
Stream In 35
Stream Out 42
PROPERTYDEFINITIONS statement
DEF data map 390
R

record description 431
  length 431
  path types 443
record types 431
reference libraries
  translating Stream 48
  using with CDL In 186
regular wiring, DEF data map 405
resizing Design Framework II designs 24
right arrow in syntax 15

S

scaling Design Framework II designs 24
si.env file 221
si.log file 239
simrc file 221
SKILL file, command option
  CIF In 155
  CIF Out 170
  Stream In 65
  Stream Out 105
SKILL functions
  general information
    Applicon, Export 279
    Applicon, Import 277
    CIF, Export 283
    CIF, Import 277
    Stream, Export 283
    Stream, Import 277
  ldtrDefRead 419
  ldtrDefWrite 421
  ldtrGenLefOutList 423
  ldtrLefRead 425
  ldtrLefWrite 427
  piCellNameMap
    Applicon, Import 271
    CIF, Import 271
    Stream, Import 271
  piCifLayerMap
    CIF, Import 273
  piLayerMap
    Applicon, Import 272
    Stream, Import 272
  pipoErrShapesHandler
    Applicon, Export 283
  Applicon, Import 277
  CIF, Export 283
  CIF, Import 277
  Stream, Export 283
  Stream, Import 277
  poCellNameMap
    Applicon, Export 280
    CIF, Export 280
    Stream, Export 280
  poCifLayerMap, CIF, Export 281
  poParamCellNameMap
    Applicon, Export 285
    Stream, Export 285
  textFieldMap
    Applicon, Import 276
    CIF, Import 276
    Stream, Export 282
    Stream, Import 276
  SKILL functions, syntax conventions 15
  slash in syntax 15
  special wiring
    DEF data map 409
  SPICE
    differences from CDL 583
  starting
    all translators 21
    CDL In 184, 209
    CIF In 158
    CIF Out 173
    nino cdlin 189
    pipo cifin 159
    pipo cifout 174
    pipo strm in 77
    pipo strmout 121
    pipo strmtechgen 77
    Stream In 76
    Stream Out 120
  Stream format 430
  differences from Design Framework II 82, 127
  font equations 33
  structure definition 50
  stream syntax 449
  Stream, Export
    ASCII dump file example 134
    error and log messages 130
    files
      input and output 91 to 93
      mapping 36
      PIPO.LOG 130
      SKILL 279
Design Data Translator's Reference

template file 96
font mapping equations 33
form 95
interpreting units 128
mapping
cell names 37
cell names with SKILL 280
fonts 41
fonts with SKILL 282
layer names 37
pCell names 285
properties 42
mapping files, syntax rules 36
output options 99
SKILL
pioErrShapesHandler 283
poCellNameMap 280
poParamCellNameMap 285
textFontMap 281
starting with pipo 121
starting with the menu command 120
translating
arrays 108
cell hierarchy 115
data types 128
illegal shapes 283
path types 127
pins 44, 102
reference libraries 49 to 50
Stream, Import
error and log messages 84
files
input and output 54 to 56
mapping 26
PIPO.LOG 85
SKILL 270
technology file 55
template file 79
font mapping equations 33
form 57
mapping
cell names 27
cell names with SKILL 271
fonts 32
fonts with SKILL 276
layer names 29
layer names with SKILL 272
properties 35
mapping files, syntax rules 26
output choices 60
SKILL
pioCellNameMap 271
plLayerMap 272
pioDisplay 270
textFontMap 276
SKILL, pioErrShapesHandler 277
starting with pipo 77
starting with the menu command 76
translating
angles 82
arrays 67
cell hierarchy 72
data types 83
illegal shapes 277
path types 83
record types 84
reference libraries 51 to 52
strong-connect pins 380
structure definitions (Stream format) 50
syntax conventions 14

T

tape, moving Stream files to disk 52
technology file
producing with Stream In 55
Template 211
template file
command option
CDL Out 211
CIF In 152
CIF Out 166
Stream Out 96
description
Stream In 54
Stream Out 91
preparing for
CDL In 190
CDL Out 220
CIF In 159
Stream In 79
sample 190, 221
terminals, map to cell pins 380
textFontMap 276
Applicon, Import 276
Stream, Export 281
Stream, Import 276
translating
cell hierarchy
CIF In 158
Stream In 72
Stream Out 115
  global nodes
  CDL In 202
  CDL Out 235
illegal shapes
  Applicon, Export 283
  Applicon, Import 277
  CIF, Export 283
  CIF, Import 277
  Stream, Export 283
  Stream, Import 277
primitive devices
  CDL In 196
  CDL Out 233
translators
  starting 21
  types of 18

U
  undefined function error 45
  UNIT statement, LEF data map 324
  user units (Stream Out), interpreting 128

V
  Verilog In 19
  vertical bars in syntax 14
  VIA statement
    LEF data map 355

W
  weak-connect pins 381